Trade-offs of Heuristic Vs. Rigorous Algorithms in Text Mining

Andrew Matheny

Problem Report submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Tim Menzies, Ph.D., Chair
James Mooney, Ph.D.
Tim McGraw, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2010

Keywords: clustering, dimension reduction, kmeans, genic, canopy, pca, fastmap, tfidf,
text mining, 20newsgroups, bbc

Abstract

Trade-offs of Heuristic Vs. Rigorous Algorithms in Text Mining

Andrew Matheny

This research assesses the cost of using heuristic methods in the field of text mining. Previous research has shown many of the formal non-heuristic algorithms to be NP-hard with positive results only in small domains. Given the massive scale when dealing with unstructured textual data, these algorithms prove to be impossible with large datasets with many dimensions. Many heuristic approximations have been proposed that drastically improve run-times, but give a result of less accuracy than the rigorous methods.

Throughout the research, we evaluate the trade-offs of heuristic methods to their more computationally complex alternatives. We focus on algorithms for document clustering and dimensionality reduction, and provide results detailing the run-times and cluster validity of each clusterer and reducer on their own, in addition to the run-times and validity scores of each clusterer and reducer combined. Our findings indicate that the cost of these approximations are as great as they first appear and that effective results can be achived using much less effort.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the 21st century, the field of program comprehension research began to focus on applying various information retrieval (IR) techniques (e.g. text mining, LSI, knowledge-based NL understanding) to software and its associated text collections (manuals, source, design documents, etc.) When this happened, despite known problems with scalability, these exhaustive IR semantic approaches became the standard. With the massive outbreak of the web, the amount of data available on which to do computations has grown exponentially. For this reason, heuristic approaches have acquired a new importance.

Today, while there have been a few looks into these heuristic algorithms, most of the research is still focused on the original exhaustive methods which have been shown to be NP-complete. Given that these heuristic methods have astounding benefits in terms of execution time due to their heuristic nature, there has yet to be a comparative study looking at the cost-benefit (or run-time to accuracy) ratio of heuristic algorithms vs their exhaustive ancestors.

The need for this analysis has arose out of a NARA (National Archives and Records Administration) funded project on the future of long-term data retention. Our toolkit and user interface,

Hamlet (see 3.1), was developed primarily for the area of technical document comprehension, specifically the STEP/EXPRESS schema (see 2.2.4). The goal being to answer the question, "What other designs are similar to this one?". Because of the massive search space of STEP designs and their associated textual project information (documentation, web pages, documentation, etc.), highly scalable algorithms were a must. Scalability however, must come at a reasonable price, that is, the loss of accuracy. Determining the cost of this scalability is the primary motivation of this research.

1.2 Contributions of This Document

This report has made the following contributions to both the literature and practice, with respect to the corpora used in this study:

- Determined empirically, the loss of accuracy when using heuristic clustering and dimensionality reduction algorithms compared to the exhaustive alternatives
- Determined empirically, the benefits of scalability when using heuristic clustering and dimensionality reduction algorithms compared to the exhaustive alternatives
- Determined empirically, the effects of algorithm parameters to the trade-off of scalability and clustering accuracy when using heuristic clustering and dimensionality reduction algorithms compared to the exhaustive alternatives
- Provided recommended algorithms and parameters to optimize the scalability/accuracy trade-off in various domains

The results of this study have shown that it is possible to greatly reduce the cost, in terms of runtime, of the IR methods we have examined with only a relatively small loss in accuracy. Different uses of these methods have varying tolerance for acceptable performance loss. We have found

that in order to optimize a solution for a specific domain, certain combinations of clusterers and reduction methods work better than others. In addition to using different combinations of reduction methods and clustering algorithms, using different values for the parameters of these algorithms can also have drastic effects on both run-time and accuracy that will need to be considered for the specific application of these methods. In general, we have found that the cost of the exhaustive methods doesn't always equal the benefits to accuracy.

1.3 Structure of This Document

The outline of the remaining chapters is as follows:

- Chapter 2 describes the necessary background information on the techniques used in this research along with other standard techniques in the field.
- Chapter 3 details the tools created as part of this research.
- Chapter 4 explains the experimental methods used, the datasets the experiment was run on, and includes an assessment of the outcomes.
- Lastly, chapter 5 summarizes the findings of this work and charters the path of future investigations.

Chapter 2

Background and Related Work

2.1 Text Mining

2.1.1 Overview

Generally speaking, text mining is the process of extracting interesting and usable information from bodies of text, stored in an unstructured format. It has been estimated that approximately 80% of information is stored as unstructured text [10] which, if examined properly, could yield new patterns of data previously unknown. The basic process typically calls for taking as input, a collection of text documents, effectively processing these documents into a storage mechanism (data structures, databases, file structures, etc.), examining the stored documents computationally, and lastly outputting some new information previously unknown about the set of documents. The field has a basis ranging from machine learning, data mining, artificial intelligence, information retrieval, statistics, natural language processing and linguistics but does not necessarily have to employ all of these sub-fields at once. Common tasks used construct this new information include document clustering and term/dimensionality reduction which are examined in this thesis, but can also include, concept location [26], text categorization [28], part-of-speech tagging [13], and text summarization [1].

2.1.2 Fundamental Concepts

Documents and Terms

At the core of any text mining task is the notion of documents and their terms. Documents, in this case, are the set of all term collections in a given corpus. A document can represent numerous types of unstructured text such as company memos, emails, news articles, source code, legal documents, etc. The terms are then the set of all unique words that appear in these documents. Heaps' Law tells us that the growth rate of the term space, or vocabulary, is approximately equivalent to the square root of the total number of words in the document set [11]. While this does mean that vocabulary grows a sub-linear rate with respect to the number of occurrences, the size of the vocabulary is still large enough to cause scalability concerns.

The Vector Space Model

The vector space model (VSM) is a algebraic construct used for nearly every text mining algorithm. It was first introduced by Gerald Salton at Cornell University in the 1960s [27] and has been a cornerstone of IR in text (and later text mining) ever since. The VSM defines each document as a vector in the space of all terms where each dimension refers to a specific term. Each value stored in the document vector represents the rate of occurrence of the corresponding term, commonly referred to as the term weight. While a number of computations can be used for the term weight the most common is the term frequency-inverse document frequency model (see section 2.1.2 below).

This approach allows the ability to produce similarity scores of the likeness of one document to another using common vector difference/similarity functions such as cosine similarity. Additionally, it provides a framework for more advanced methods such as latent semantic indexing (LSI) and principal component analysis (PCA). However, there are drawbacks to the VSM such as high dimensionality of the term space, the loss of word ordering, and no representation of the similarity

between words with similar meanings but different words.

TF*IDF

In order to perform mathematical operations and algorithms on documents and the text that they contain, we must first transform them into a representative mathematical object. The standard representation of a document is a vector in the space of all available terms. For example, the phrase:

$$\begin{gathered} \textit{The quick brown dog was very} \\ \textit{quick, very brown, and very dog like.} \end{gathered} \tag{2.1}$$

Will be turned into a vector which looks something like this:

$$\textit{Phrase} = [1\,2\,2\,2\,1\,3\,1\,1] \tag{2.2}$$

with each index of the above vector corresponding the a dimension which comes from the term list (in this case, the dimensions are the, quick, brown, dog, was, very, like)

Tf*Idf is shorthand for "term frequency times inverse document frequency." This calculation models the intuition that jargon usually contains technical words that appear a lot, but only in a small number of paragraphs. For example, in a document describing a space craft, the terminology relating to the power supply may appear frequently in the sections relating to power, but nowhere else in the document.

Calculating Tf*Idf is a relatively simple matter:

- Let there be $Words$ number of documents;
- Let some word I appear $Word[I]$ number of times inside a set of $Documents$;
- Let $Document[I]$ be the documents containing I.

Then:

$$Tf * Id = Word[i]/Words * log(Documents/Document[i])$$

2.1.3 Linear Preprocessing Tricks

Aside from the more intensive and complex reduction methods described later on, there are also several known preprocessing tricks that can reduce the size of the set of all possible terms. Three of these are listed below.

Tokenization

In HAMLET's parser, words are reduced to simple tokens via (e.g.) removing all punctuation remarks, then sending all upper case to lower.

Stop lists

Another way to reduce dimensionality is to remove "dull" words via a *stop list* of "dull" words. Figure 2.1 shows a sample of the stop list used in HAMLET. Figure 2.1 shows code for a stop-list function.

```
a          about    across   again      against
almost     alone    along    already    also
although   always   am       among      amongst
amongst    amount   an       and        another
any        anyhow   anyone   anything   anyway
anywhere   are      around   as         at
...        ...      ...      ...        ...
```

Figure 2.1: 24 of the 262 stop words used in this study.

Stemming

Terms with a common stem will usually have similar meanings. For example, all these words relate to the same concept.

- CONNECT
- CONNECTED
- CONNECTING
- CONNECTION
- CONNECTIONS

Porter's stemming algorithm [23] is the standard stemming tool. It repeatedly replies a set of pruning rules to the end of words until the surviving words are unchanged. The pruning rules ignore the semantics of a word and just perform syntactic pruning (e.g. Figure 2.2).

RULE			EXAMPLE		
ATIONAL	->	ATE	relational	->	relate
TIONAL	->	TION	conditional	->	condition
			rational	->	ration
ENCY	->	ENCE	valency	->	valence
ANCY	->	ANCE	hesitancy	->	hesitance
IZER	->	IZE	digitizer	->	digitize
ABLY	->	ABLE	conformably	->	conformable
ALLY	->	AL	radically	->	radical
ENTLY	->	ENT	differently	->	different
ELY	->	E	vilely	->	vile
OUSLY	->	OUS	analogously	->	analogous
IZATION	->	IZE	vietnamization	->	vietnamize
ATION	->	ATE	predication	->	predicate
ATOR	->	ATE	operator	->	operate
ALISM	->	AL	feudalism	->	feudal
IVENESS	->	IVE	decisiveness	->	decisive
FULNESS	->	FUL	hopefulness	->	hopeful
OUSNESS	->	OUS	callousness	->	callous
ALITY	->	AL	formality	->	formal
IVITY	->	IVE	sensitivity	->	sensitive
BILITY	->	BLE	sensibility	->	sensible

Figure 2.2: Some stemming rules.

Porter's stemming algorithm has been coded in any number of languages[1] such as the Perl *stemming.pl* used in this study.

[1] http://www.tartarus.org/martin/PorterStemmer

2.2 Technical Document Comprehension

2.2.1 Overview

Technical document comprehension (also referred to as program comprehension) focuses on learning patterns and hidden details in collections of technical documents. While the majority of work being done in technical document comprehension is based on evaluating source code, other types of documents can be explored as well. Legal documents [25], UML [14], and even HTML [21] have all been explored using standard IR and text mining techniques. The potential benefit of focusing on technical documents is utilizing the inherent structure that is found in them to build previously unknown relationships. Ideally, once these new relationships are combined with existing knowledge gained through text mining, results can greatly improve. In the next few sections I will examine a few existing tools for technical document comprehension in the realm of software engineering.

2.2.2 HIPIKAT

Hipikat [4] by Cubranic and Murphy is an attempt at a collective project memory, developed as plug-in for the Eclipse development environment. It works by bringing in textual information from CVS repositories, change requests (issue reports, feature requests, etc), communication channels (forum posts, emails, etc.), design documents, and other software engineering artifacts found on the projects website. Hipikat analyses these documents and can infer links between artifacts pertaining to a similar subject. For instance, if a bug was marked as fixed in the issue tracker and within a given time-frame, a CVS check-in was submitted, a possible link can exist between these two artifacts. As new artifacts are submitted to Hipikat, they are examined for potential links and the entire collection is updated. What is created is a network of linked artifacts which, when combined with a similarity analysis of the text in each node, can be used to provide fine-tuned artifact

recommendations. When using the tool, a developer can either perform a keyword search through the Hipikat database or query Hipikat by selecting an artifact in the IDE, such as a source code file, CVS submission, or bug report. In both instances, a list of suggested articles is presented which can then be used for further Hipikat queries, or opened for viewing. While Hipikat has been shown to aid newcomers to a project with grasping a general understanding of a codebase, its downfall is the scalability issue. Hipikat uses LSI [5] to perform similarity tests between documents which is costly in both space and time. In addition to performance issues when working with larger collections, there are also concerns with the quality of the recommendations as larger collections tend to be more heterogeneous in nature.

2.2.3 Dora the Program Explorer

Dora is, like Hipikat, also an Eclipse plug-in that tackles the same problem of finding similar code artifacts based on an initial artifact. Unlike Hipikat which uses communication channels, issue reports, etc. to build its searchable repository, Dora only uses Java methods. As a result of only using methods, the relationships between nodes in Dora is of a more syntactic nature since they are generated from call graphs, where a relationship exists between two methods if one calls the other. Using only call graphs in large software projects is not the most effective approach given that there are many neighboring functions and many of those that are of little reference to the seed method. This is where the term similarity between methods is used to help trim down the neighborhood. Dora uses a series of similarity comparisons including TFIDF 2.1.2 that is then combined using a linear model to generate a similarity measure between methods that is then used to build a relevant neighborhood. When computing a similarity measure between two methods, Dora will consider if the method is a binary method taken from a library combined with TFIDF similarity between method names, identifiers used in the method body, and the natural language in code comments. To help reduce these term sets even further, Dora employs some of the preprocessing tricks mentioned in section 2.1.3 such as stemming.

2.2.4 STEP/EXPRESS

STEP (Standard for the Exchange of Product model data) is an ISO specified [15] language used to represent product manufacturing information. The goal a mechanism capable of describing production data from the entire life-cycle of a product. independent of any specific system or methods the product employs. This yields transferability between existing software and machine systems. By nature, this makes it useful not only for a neutral file format, but also as a starting point for creating and sharing product databases and archiving. All STEP files must conform to a schema defined in the EXPRESS language. Essentially, EXPRESS is a data modeling language that defines the structure of STEP documents and STEP documents are EXPRESS schemas instantiated. The ISO has defined a number of these schemas, known as *application protocols*. See Figure 2.3 for the list of currently defined APs.

In theory, there is nothing stopping STEP/EXPRESS from recording and storing all aspects of a project. In many ways, STEP/EXPRESS is as expressive as other technical document standards (e.g. UML). STEP/EXPRESS offers a generic method for storing part-of and is-a information, constraints, types, and the rules associated with a technical document. However, in practice, the theoretical potential of STEP/EXPRESS is not realized for the following reasons.

Heterogeneity

The reality of archival systems is that STEP/EXPRESS documents are stored *along side* a much larger set of supporting documents in multiple formats. Given this, if we can learn how to understand large heterogeneous collections that include STEP/EXPRESS knowledge as well as numerous other products in a wide variety of formats, it would be possible to reason and learn from a very wide range of data.

A majority of the work has focused on the creation of cached sets of EXPRESS schemas. Forty such *application protocols* (AP) have been defined [16] including AP-203 (for geometry) and AP-

213 (for numerical control). The list of currently defined application protocols is very extensive (see Figure 2.3). These APs are the cornerstone of STEP tools: the tools offer specialized support and screen import/export facilities for the APs. While much effort went into their creation of these APs, very few have been stress-tested in the information systems field. That is, the majority of these APs have been *written* more than they have been *read* (exceptions: the above-mentioned AP-203 and AP-213 are frequently used and reused in 21^{st} century CAD/CAM manufacturing processes),

Perhaps because of the relative immaturity of the APs, current CAD/CAM tools offer limited support for the STEP APs. While most tools support geometry (AP-203), the support for the other APs in Figure 2.3 is minimal.

Limited Historical Use

For all the above reasons, highly structured technical documents in formats like STEP/EXPRESS are in the minority in the archival systems we have examined. We are aware of large STEP/EXPRESS repositories but these are often inaccessible for a variety of reasons.

While this situation might change in the future (e.g. if all the above issues were suddenly fixed and all organizations switch to using highly structured technical documentation), the historical record would still be starved for large numbers of examples.

2.3 Term Reduction

2.3.1 Principal Component Analysis (PCA)

Principal component analysis (PCA) is a mathematical method rooted in the singular value decomposition of a matrix to infer correlation from a large set of previously uncorrelated variables [17]. The new set of variables is of a much smaller size which could prove useful for reducing run-times

AP	area
201	Explicit Drafting
202	Associative Drafting
203	Configuration Controlled Design
204	Mechanical Design Using Boundary Representation
205	Mechanical Design Using Surface Representation
206	Mechanical Design Using Wireframe Representation
207	Sheet Metal Dies and Blocks
208	Life Cycle Product Change Process
209	Design Through Analysis of Composite and Metallic Structures
210	Electronic Printed Circuit Assembly, Design and Manufacturing
211	Electronics Test Diagnostics and Remanufacture
212	Electrotechnical Plants
213	Numerical Control Process Plans for Machined Parts
214	Core Data for Automotive Mechanical Design Processes
215	Ship Arrangement
216	Ship Molded Forms
217	Ship Piping
218	Ship Structures
219	Dimensional Inspection Process Planning for CMMs
220	Printed Circuit Assembly Manufacturing Planning
221	Functional Data and Schematic Representation for Process Plans
222	Design Engineering to Manufacturing for Composite Structures
223	Exchange of Design and Manufacturing DPD for Composites
224	Mechanical Product Definition for Process Planning
225	Structural Building Elements Using Explicit Shape Rep
226	Shipbuilding Mechanical Systems
227	Plant Spatial Configuration
228	Building Services
229	Design and Manufacturing Information for Forged Parts
230	Building Structure frame steelwork
231	Process Engineering Data
232	Technical Data Packaging
233	Systems Engineering Data Representation
234	Ship Operational logs, records and messages
235	Materials Information for products
236	Furniture product and project
237	Computational Fluid Dynamics
238	Integrated CNC Machining
239	Product Life Cycle Support
240	Process Planning

Figure 2.3: STEP Application Protocols.

of complex and time-demanding computations. PCA is is distinct in that is the optimal linear transformation for keeping the subspace which has the largest variance. For example, in figure 2.3.1, PCA is performed on the two dimensional plot on the left and the result shows the new one dimensional plot which best diplays the variance in the data.

The resulting subspace has been shown to be highly beneficial when clustering with K-Means due to the fact that it is identical to the cluster centroid subspace specified by the between-class scatter matrix, which is where the global solution of K-Means lies [6]. Despite this, PCA is the most computationally complex reduction algorithm being tested in this study and is not usable in most applications due to its non-scalability.

2.3.2 FastMap

The general goal of FASTMAP is to project items in a n dimensional to a d dimensional space, with $n > d$. The basis of each reduction is using the cosine law on the triangle formed by an object in the feature space and the two objects that are furthest apart in the current (pre-reduction) space (see Figure 2.5). These two objects are referred to as the pivot objects of that step in the reduction phase (n - d total pivot object sets). Finding the optimal solution of the problem of finding the two furthest apart points is a N^2 problem (where N is the total number of objects), but this is where the heuristic nature of FASTMAP comes into play.

Instead of finding the absolute furthest apart points,
FASTMAP takes a shortcut by first randomly selecting an object from the set, and then finding the object that is furthest from it and setting this object as the first pivot point. After the first pivot point

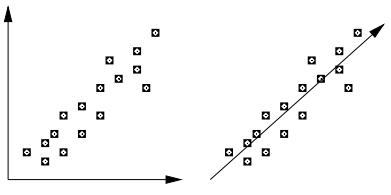


Figure 2.4: The two features in the left plot can be transferred to the right plot via one latent feature.

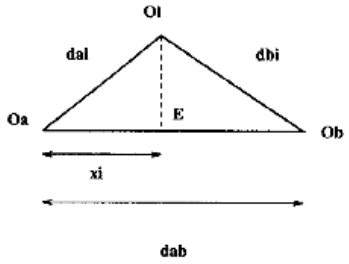


Figure 2.5: Example of using the cosine law to find the position of Oi in the dimension k

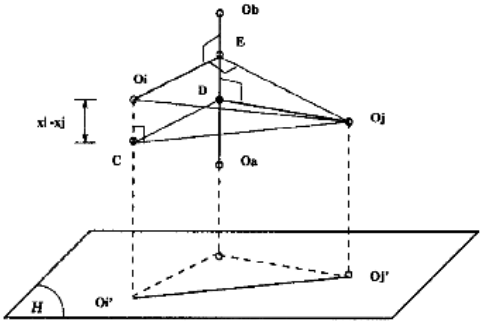


Figure 2.6: Projects of points O_i and O_j onto the hyper-plane perpendicular to the line O_aO_b

is selected, FASTMAP finds the points farthest from this and uses it as the second pivot point. The line formed by these two points becomes the line that all of the other points will be mapped to in the new n - 1 dimension space.

2.3.3 TF*IDF Term Ranking

TF*IDF term ranking is linear time approach to reducing dimensionality in text mining datasets. It is a simple algorithm created to benchmark the results of PCA and FastMap for the purposes of the study. Its basis is the idea that the TF*IDF value of a word represents that words importance in the context of the dataset. The algorithm work by summing the TF*IDF scores of each term across all documents and then returning the n terms with the highest summed values. See figure 2.7 for a sample of key terms returned by TF*IDF term raking on one of our STEP datasets.

```
cartesian_point   type        oriented_edge   subtype         entity
label             sizeof      self            query           select
name              direction   edge_curve      where           wr1
text              real        set             not             description
for               rule        items           typeof          supertype
...               ...         ...             ...             ...
```

Figure 2.7: 30 of the 100 key terms found in a STEP dataset using TF*IDF ranking

One drawback to this method is that it requires a much larger value for n to be used effectively. The value for n must be large enough to encompass any term which later will be used to query the dataset later. Also, as the value of n decreases, the dataset will become increasingly homogeneous and many documents will be represented as nearly identical to completely different documents within the collection. However, the drastic difference in run-time of this method is very appealing, especially given the non-scalable nature of PCA and LSI.

2.4 Document Clustering

2.4.1 K-Means (elementary)

K-Means is a clustering algorithm that, when given a dataset of unidentified objects, it will group those items into k groups based on some given similarity measure. The algorithm is described in Figure 2.8. For an example of the algorithm in operation, see Figure 2.9.

While k-means may be sufficiently accurate, there are significant drawbacks. Most notably is the speed (or lack thereof). Due to the k-means algorithm having to compute distances from every item to every cluster. In situations where the cosine similarity distance measure is used, computing the distance between points can be an expensive operation (this is another place dimensionality reduction helps out). In recent tests comparing clustering algorithm run-times, k-means was found to be up to 500 times slower than another algorithm, GENIC, which we discuss further down.

Another problem with k-means is determining what value of k should be used. Note the usability issues with requiring a user to pre-specify k: isn't this the kind of tedious detail that the

- i=0
- Partitioning the input points into k initial sets, either at random or using some heuristic data.
- Repeat unitl ($i \leq maxIterations$ or no point changes set membership)
  - Calculates the mean point, or centroid, of each set or cluster.
  - Constructs a new partition, by associating each point with the closest centroid.
  - Recalculate the centroids for the newly partitioned cluster
  - i = i + 1

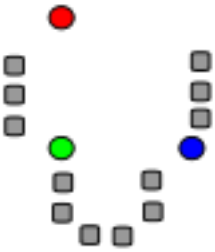
Figure 2.8: K-Means algorithm. See Figure 2.9 for an example of this algorithm running in practice.

computer should be telling us?

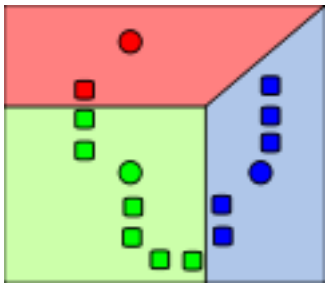
2.4.2 K-Means (canopied)

The canopy clustering algorithm is an unsupervised clustering algorithm built on top of K-Means. Its primary claim is the increased speed when working with large datasets. With large enough datasets K-Means can be utterly useless. Canopy clustering allows us to use these algorithms with a marginal loss of accuracy for magnitudes of improvements in run time. Its processes is based on two key steps. First, in a preprocessing step partition items into canopies based on similarity. This first step has to be done with a minimalist approach in order to keep the operation lightweight. To facilitate this, a very cheap distance measure is used. Typically this distance metric is tied to the domain. In this study, our cheap distance metric was number of similar words in two documents. After this set of canopies is generated, K-Means continues as normal with one exception, only use the more expensive similarity measure on items belonging to the same canopy. The basic steps of the algorithm are listed below.

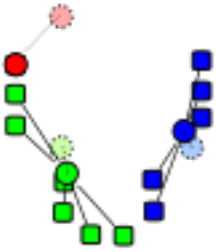
- Begin with a set of points and remove one at random. Create a Canopy containing this point and iterate through the remainder of the point set.



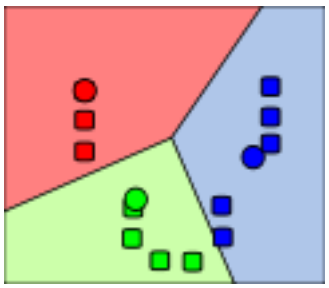
(a) Randomly select cluster centroids



(b) Assign each object to it's nearest centroid



(c) Compute new centroids by using the mean value from the centroids picked in step (b)



(d) Repeat steps (b) and (c) until the convergence criteria has been reached

Figure 2.9: Figures (a) through (d) above illustrate the major steps of the kmeans algorithm

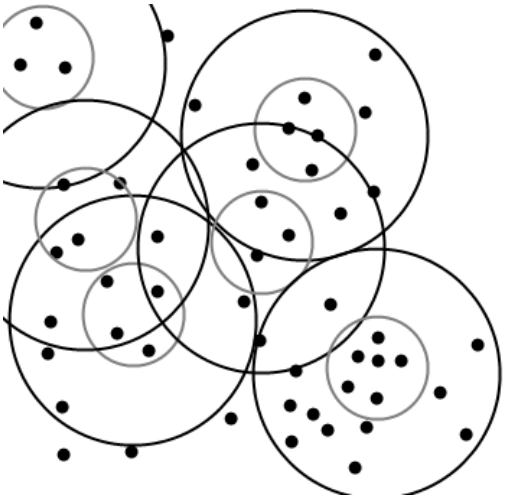


Figure 2.10: The darker circle represents all points in a given canopy, points in the smaller circles cannot be used as a new canopy center.

- At each point, if its distance from the first point is $<$ T1, then add the point to the cluster. If, in addition, the distance is $<$ T2, then remove the point from the set. This way points that are very close to the original will avoid all further processing. The algorithm loops until the initial set is empty, accumulating a set of Canopies, each containing one or more points.

2.4.3 Genic

GENIC is a generalized incremental clustering algorithm developed by Gupta and Grossman [12] that provides potentials for large improvements in scalability over K-Means. Since GENIC was designed with streaming data in mind, it only has a single pass through the data to work with. Because of this, it scales linearly, which is a requirement when dealing with large corpora. By using stochastic methods, GENIC can be given an initial k equal to the number of items (each item is its own clusters) and prune away unlikely clusters with each generation, giving a realistically estimated value for k after the last generation. Here is how GENIC works:

1. **Select parameters**

    - Fix the number of centers *k*.

- Fix the number of initial points m.
- Fix the size of a generation n.

2. **Initialize**
   - Select m points, $c_1, ..., c_m$ to be the initial candidate centers.
   - Assign a weight of $w_i = 1$ to each of these candidate centers.

3. **Incremental Clustering** For each subsequent data point p in the stream: do
   - $Count = Count + 1$
   - Find the nearest candidate center c_i to the point p
   - Move the nearest candidate center using the formula

$$c_i = \frac{(w_i * c_i + p}{w_i + 1} \tag{2.3}$$

   - Increment the corresponding weight

$$w_i = w_i + 1 \tag{2.4}$$

   - When $Count\ mod\ n = 0$, goto Step 4

4. **Generational Update of Candidate Centers**

   When $Count$ equals $n, 2n, 3n$, ..., for every

   center c_i in the list L of centers, do:
   - Calculate its probability of survival using the formula

$$p_i = \frac{w_i}{\sum_{i=1}^{n} w_i} \tag{2.5}$$

- Select a random number δ uniformly from [0,1]. If p_i ¿ δ, retain the center c_i in the list L of centers and use it in the next generation to replace it as a center in the list L of centers.
- Set the weight w_i = 1 back to one. Although some of the points in the stream will be implicitly assigned to other centers now, we do not use this information to update any of the other existing weights.
- Goto step 3 and continue processing the input stream

5. **Calculate Final Clusters** The list L contains the *m* centers. These *m* centers can be grouped into the final *k* centers based on their Euclidean distances.

GENIC is of specific interest to HAMLET for two primary reasons, low expected run-times on large corpora and a potential ability at estimating the number of natural clusters in the collection.

- Scalability: Since GENIC was designed with streaming data in mind, it only has a single pass through the data to work with. Because of this, it scales linearly, which is a requirement if HAMLET is to scale to large corpora.
- An likely estimate for *k*: Because of GENIC's stochastic based method of removing unwanted or non-useful clusters, it has potential for use in correctly estimating a good value for *k*. By eliminating "bad stuff", GENIC can ideally identify the correct number of types of "good stuff".

Chapter 3

Tools

3.1 Hamlet

3.1.1 Motivation

A recent NSF-funded workshop highlighted current directions in long term technical document retention. While much progress was reported on:

- systems issues of handling and sharing very large data collection (e.g. SLASH)
- scalable methods of building customization views (e.g. iRODS)

there was little mention of the cognitive issues of how users might browse and synthesize data from massive data collections of technical documents.

For example, here at WVU, we are mid-way through a review of the use of STEP/EXPRESS for long term technical document retention. STEP/EXPRESS is commonly used as an inter-lingua to transfer technical data between CAD/CAM packages. Strange to say, while STEP/EXPRESS is useful for transferring and understanding technical documents *today*, it does not appear to be suitable for understanding technical documents from *yesterday*.

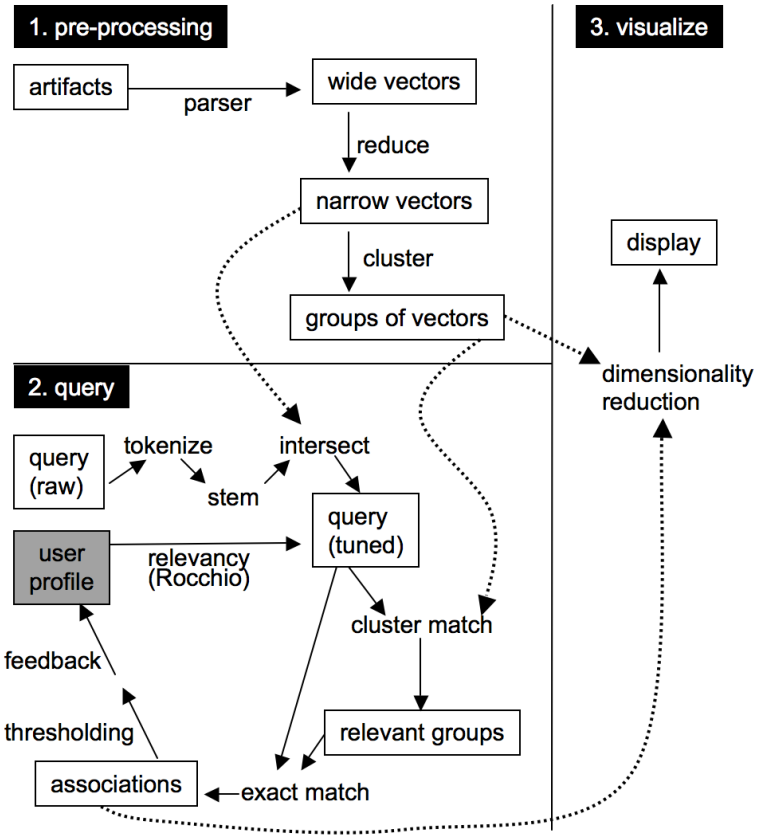


Figure 3.1: HAMLET's search for associations.

3.1.2 Overview

HAMLET's search for association has to two essential phases, and a third optional phase (see Figure 3.1). All these stages have the same goal: from a large set of possible associations, extract the small subset that (a) have occurred frequently in prior technical documents and which (b) the user will approve. Before running any query:

- Archival artifacts must be parsed into a network of nodes and edges. Each node is represented as a set of *terms* referenced in that node. This set of terms can include every term in the archive so we call these the *wide vectors*.

- The wide vectors are unwieldy to process. Hence, we run some linear time *reduction* meth-

ods to isolate the most informative columns. This produces the *narrow vectors*. Note that, when we run a query, we only use the parts of the query that appear in these narrow vectors.

- HAMLET also clusters the narrow vectors into *groups*. These groups define cliches of repeated structures.

When running a query:

- We remove irrelevant detail from the query. Linear time methods remove white space and spurious word endings. Then the query is pruned back to include just the terms seen in the narrow vectors.

- The resulting *tuned vector* is then matched to the the groups of vectors found above. For computational reasons, we run this match in two-stages. First, we find the N nearest clusters (this generates the *relevant groups*). Second, searching just within those relevant groups, we perform exact matches to find related terms.

- The *candidate matches* found from the exact match are then ranked (by the size of the overlap of the query and the terms) then pruned using thresholding (find the neighboring items in th sort with the biggest difference between them; prune the items below that largest cliff).

- The user assesses the matched queries and declares that some are "relevant" and some aren't. This builds up a session log for this user working these kinds of queries. Once this log grows beyond a certain size, it is used to refine the tuned query such that the tuning favors nodes that do not contain what the user has labeled "irrelevancies" and does contain what the user has called "relevances".

Optionally, we can visualize the results:

- The N-dimensions of the vectors are mapped down to 3 dimensions, then visualized on the screen.

3.1.3 Parsing Framework

HAMLET's language-specific subparsers comb through individual files and pull out important bits of information (entities in STEP, methods in Java). While processing individual files, these subparsers collect information about each document. Some of that information includes pointers to the parsed information and information about what entities are used by others. The HAMLET generic parsing framework provides several methods to utilize these data attributes.

The most important function of the API is the generation of the GraphXML file. This file is the intermediary between the data set and HAMLET. It contains a list of each document (vertice) and the relationships between them. Other pertinent information, such as file pointers and document statistics, is stored in the form of attributes for each document vertice. From a higher level, the collection of these pointers to files gives a view of the region of interconnected designs, giving HAMLET the ability to make its decisions and provide suggestions based on what it has already learned.

Figure 3.2: One vertice and the information associated with it.

For certain language imports, such as Java or STEP, HAMLET utilizes edge generation to determine the relationship of one design to another. For instance, if a call graph is generated on a set of Java source files, an edge can be placed between a multitude of methods and calls made to

and from them. The XML graph generated by the parsing API includes both the document vertices and the edges that connect them. This is essential for visualization purposes and provides a wealth of syntactical information.

Internally, HAMLET makes minimal assumptions about the form of the technical document:

- A document contains slots and slots can be atomic or point to other documents;.
- The network of pointers between documents presents the space of connected designs.

A generic parser class implements a standard access protocol for this internal model. By sub-classing that parser, it is possible to quickly process new documents types. Currently, HAMLET's parsers can import:

- STEP/EXPRESS
- Florida Law (XML)
- Text documents structured as follows: sub-headings within headings, paragraphs within sub-headings, sentences within paragraphs, words in sentences;
- JAVA: This JAVA import allows ready access to very large corpora of structured technical information (i.e. every open source JAVA program on the web). Hence, in the sequel, we will make extensive use of JAVA examples since that permits tests of scalability.

3.1.4 Preprocessor

HAMLET contains several components other than the UI shown below. One such com- ponent is the pre-processor, a tool used by HAMLET to generate datasets which can then be loaded into the UI allowing the user to query, rank, and visualize the documents found in the loaded dataset. The majority of all machine learning takes place within the pre-processor. This is where tasks like term frequency / document frequency gen- eration, term selection, clustering, and learner

training occurs. After being run through the pre-processor, each document within a collection is assigned a vector representation which describes what terms are present in the document and at what frequency.

3.2 Ourmine

In an effort to provide a framework that better allows for full collaboration between researchers and repeatability of experiments, Ourmine [8] was used to script the experiments. Further information about Ourmine is available in section 3.2. To extend the capabilities of Ourmine into the domain

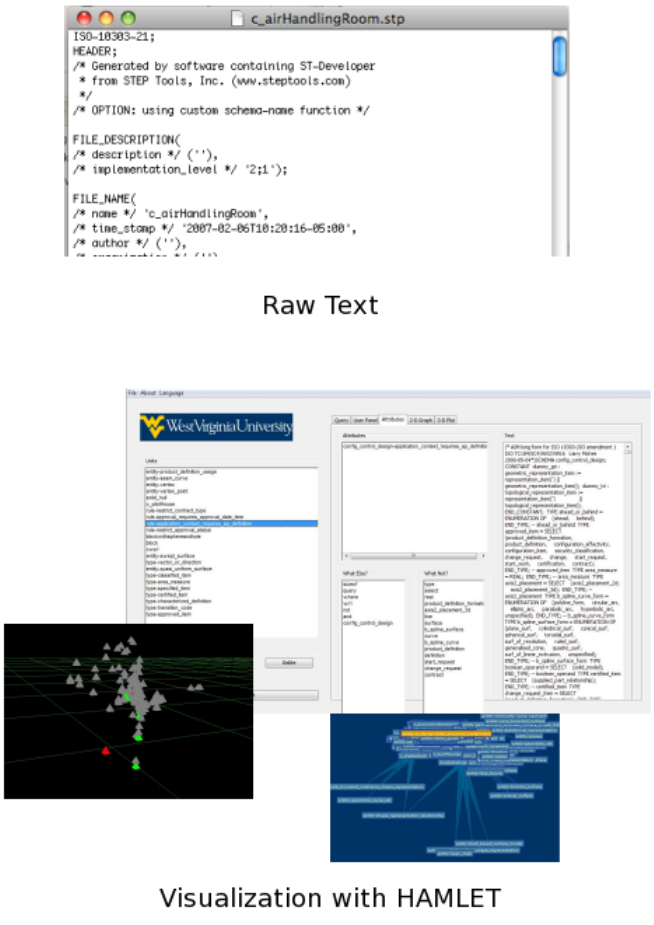


Figure 3.3: Looking at data using HAMLET has several visualization advantages

of text mining, a modified version, Textmine, was created that adds methods for term reduction, document clustering, and evaluation of document clusters into Ourmine's toolset.

Chapter 4

Laboratory Studies

This chapter describes the laboratory studies performed throughout this research. There are a total of three main experiments that assess the run-time and accuracy in the following areas: term-space reduction (section 4.2.1), document clustering (section 4.2.2), and document clustering using using all pairs of reduction methods and clusterers (section 4.2.3). In each experiments section we discuss the procedures used, run-time results, validity (accuracy) results, effects of variables, and a look at the observed trade-offs. Section 4.1 lists the basic setup of the experiments including the datasets and algorithms used, evaluation metrics, parameters being varied and parameters held constant, and some initial expectations. Lastly, section 4.2 captures the summary of all relevant findings of our research.

4.1 Design

4.1.1 Considered Algorithms

Of the six different algorithms compared in this study, there are three dimensionality reduction methods and there are three clusterers. Within each category (reduction and clustering), we have one exhaustive method and two heuristic methods. The three reduction methods being studied

are: PCA [22] (exhaustive), FastMap [7] (heuristic), and TF-IDF Ranking (heuristic). The three clustering algorithms are: K-Means [18] (exhaustive), Canopy [20] (heuristic), and Genic [12] (heuristic).

The two exhaustive algorithms (PCA and K-Means) were chosen as they provide a roughly generic representation of non-scalable clustering and dimension reduction. The remaining four were either chosen because of a mix of their astounding run-time advantages and non-existing prior comparisons in the literature.

Exhaustive/Heuristic	**Clustering**	**Dimensionality Reduction**
Exhaustive	K-Means	PCA
Heuristic	Canopy	FastMap
	Genic	TF-IDF Ranking

Table 4.1: Supervised datasets along with class values.

4.1.2 Datasets

The datasets chosen for these experiments listed in figure 4.2, fall into two categories: supervised and unsupervised. Consequently, all supervised datasets used are standard and freely available text mining collections primarily in the form of news articles and newsgroup posts. Our unsupervised datasets were created in-house from collections of two STEP/EXPRESS application protocols that are highly syntactic in nature and help fulfill our primary motivations described in section 1.1. See figure 4.2 for a listing of datasets and relevant statistics.

Documents	AP203	AP214	BBC	BBCSport	ngBias3	ngBias8	ngBal3	ngBal8
D documents	484	1373	2224	737	1500	2395	1499	3999
T terms	1103	3050	9635	4613	8631	9826	8158	14984
Mean document length	143	164	130	104	116	87	91	90
Natural Classes	N/A	N/A	5	5	3	8	3	8
Stemming used	-	-	x	x	x	x	x	x
Stop-words removed	-	-	x	x	x	x	x	x

Table 4.2: Statistics on our datasets.

Dataset	Natural classes
BBC	business(509), entertainment(386), politics(417), sport(511), tech(302)
BBCSport	athletics(100), cricket(124), football(265), rugby(147), tennis(57)
ngBias3	graphics(136), hockey(591), windows(587)
ngBias8	atheism(704), autos(202), crypt(204), hockey(205), mac(202), mideast(202), space(204), xwindows(205)
ngBal3	graphics(463), hockey(459), mideast(453)
ngBal8	atheism(471), autos(463), crypt(467), hockey(461), mac(466), mideast(470), space(461), xwindows(469)

Table 4.3: Supervised datasets along with class values.

All datasets other than AP203 and AP214 (both of which were generated for this project) were found at the Machine Learning Group of University College Dublin website [2]. The BBC and BBCSport datasets were first introduced in [9]. BBC is made up news articles taken from the BBC website from 2004-2005, similarly BBCSport is made up of news articles from the BBCSport website during the same time period. Each contains 5 natural classes that identify the articles topical area. The remaining datasets are all subsets of the 20Newsgroups dataset introduced in [19] and currently residing in the UCI KDD Archive [3]. Both ngBias3 and ngBias8 come datasets with uneven class frequencies. In ngBias3, two classes each represent approximately 45% of the documents, while the other only represents approximately 10%. In ngBias8, 7 classes each represent approximately 9% of the documents, while the remaining class represents approximately 33%. In both ngBal3 and ngBal8 the distribution of class frequencies is approximately equal. See figure 4.3 for exact distributions. All of the 20Newsgroups subsets used in this study have well-separated classes, meaning they were chosen in such as way as to minimize overlap between classes.

Both the 20Newsgroups and BBC datasets were chosen because of their previous use in the literature that allows us to draw a comparison to a known benchmark. As of May 2009, the 20Newsgroups introductory paper [19] has been cited over 200 times according to CiteSeerX. While such

a large claim cannot be made by the BBC datasets, they have only been around a few years and look promising for future collaboration. While the two STEP datasets cannot provide any external validity to our findings, they do add an additional perspective due to their drastic differences in content.

4.1.3 Metrics

This section describes the metrics that are used to evaluate the algorithms listed above. These values can also be considered the dependent variables of the experiments. They were all chosen with the goal of providing a method of evaluation that is applicable to both supervised and unsupervised datasets since the ultimate working environment of hamlet (section 3.1) may be in either of these domains. To eliminate bias from running on different hardware, all values will be listed as ratios to the exhaustive methods. In addition to removing hardware bias, this will also focus on the relative values; these will be of higher importance than the raw values since comparison between methods is what we are most concerned with.

Run-times

For obvious reasons, run-time will be a primary metric for evaluating both the clustering and dimension reduction algorithms. Run-times will be measured in seconds and then converted to a ratio of the exhaustive methods.

Internal Similarity

Internal similarity (equation 4.1.3) will be used for measuring the validity of both clustering and reduction. A benefit of this measure is that it can be applied to both unsupervised and supervised datasets since no information about the correct classification is required. This allows this measure to provide us with additional insight into our generated datasets (see STEP datasets in section 4.1.2), allowing us to see whether they are of any further value to the development of hamlet.

$$iSim_i = \sum_{d \in C_i} \sum_{d' \in C_i} \frac{cos(d,d')}{n_i^2} \; iSim = \sum_i \frac{n_i}{N} iSim_i$$

Figure 4.1: Equation for internal similarity

$$eSim_{ij} = \sum_{d \in C_i} \sum_{d' \in C_j} \frac{cos(d,d')}{n_i n_j} \; eSim = \sum_i \frac{n_i}{N} eSim_{ij}$$

Figure 4.2: Equation for external similarity

Internal similarity is primarily used a means of assessing the results of a particular clustering. We'll use it for reduction by looking at the validity of the clusters generated using each of the reduction methods. Essentially, internal similarity is the measure of how similar items that belong to the same cluster are to each other. Since in a good clustering, we expect items in the same cluster to very similar to each other, a good value for internal similarity is one that approaches that maximum value of 1.0.

External Similarity

External similarity (equation 4.1.3) will be used for measuring the validity of both clustering and reduction. A benefit of this measure is that it can be applied to both unsupervised and supervised datasets since no information about the correct classification is required. This allows this measure to provide us with additional insight into our generated datasets (see STEP datasets in section 4.1.2), allowing us to see whether they are of any further value to the development of hamlet. External similarity is primarily used a means of assessing the results of a particular clustering. We'll use it for reduction by looking at the validity of the clusters generated using each of the reduction methods. Essentially, external similarity is the measure of how similar items that belong to different clusters are to each other. Since in a good clustering, we expect items in different clusters to not be very similar to each other, a good value for external similarity is one that approaches that minimum value of 0.0.

Similarity Loss = *External Similarity - Internal Similarity*

Figure 4.3: Equation for similarity loss

Similarity Loss

SimilarityLoss is an original measure developed for the research. It is a way of combining the results of relative ratios of internal and external similarity for different algorithms and datasets. After the computing the exact values of internal and external similarity for each of the algorithms, we express them each as a ratio of their value to the value of most exhaustive algorithm. The result is a value which indicates the relative gain or loss of whatever measure the ratio is generated on. When dealing with relative internal similarity scores, good values are those that have relative values greater than one (meaning they out-preformed the exhaustive method). With external similarity, a good value is one with a relative value less than one (also meaning a higher level of performance compared to the exhaustive method). Our studies have shown that often times a method will show an improvement in one of the similarities with a dis-improvement in the other. Similarity loss is a measure of how much overall similarity (both internal and external) is lost or gained using a particular method. It is defined below in equation 4.1.3 as simply the difference between external similarity and internal similarity.

Cluster Purity

Cluster purity is a measure that can only be used on supervised datasets. Intuitively, it is a measure of the homogeneity of class values within each cluster. Since purity requires an initial pairing of each item with a class label (meaning it is a supervised measure), it cannot be used on our STEP datasets (section 4.1.2). Using supervised datasets and not evaluating the results using the added information of class labels would be overlooking an insightful component of information. Values for purity range from 0.0 - 1.0 with higher values telling of a *morepure* clustering. Like the other measures used in these experiments, purity for each dataset and algorithm will be scaled by the

purity value of the most exhaustive method for that dataset.

Variable of	Name	Description	Values	Rational
General Reduction	d	The number of dimensions being reduced to	3, 15, 25, 50, 100, 200	We reduce all the way down to 3 dimensions to examine 3-d visualizations of the different treatments. values greater than this are used to test the run-time growth of the treatments.
General Clustering	k	The number of clusters requested	3, 5, 8, 15, 40, 75	The levels 3, 5, and 8 were chosen because each corresponds to the number of classes in the various supervised datasets. The others were chosen to fit our generated datasets and to show run-time growth with larger k.
KMeans Clustering	$max\_iterations$	The number of recursive step KMeans will take until it accepts the current clustering	500	Set as recommended in the literature [6]
Genic Clustering	c	The number of centers or clusters	{set to the same values of k above}	Used as a replacement for k (listed above).
	m	The number of initial points	{set to the same values of k above}	Set to the value of c. To keep consistency between all of the clusterers, this value was set to stay constant with c and k.
	n	The number of generations	$\frac{size(dataset)}{15}$	Set to change with whatever dataset is used. This keeps the number of passes constant among all treatments.
Canopy Clustering	$tight\_threshold$	Using the cheap distance metric of count of similar terms. If two items are more similar than this they must both be canopy centers.	10	Using Early experimentation showed the number documents that have more than 10 similar terms occur infrequently enough to provide an efficient number of canopies.
	$loose\_threshold$	Using the cheap distance metric of count of similar terms. If an item is this similar or more to a canopy center, it is assigned to that canopy.	4	Using Early experimentation showed the number documents that have 4 or more similar terms occur at a frequency that provides an efficient distribution of canopy sizes.
	$max\_iterations$	The number of recursive steps KMeans will take until it accepts the current clustering	500	Set as recommended in the literature [6]

Table 4.4: Listing of independent variables and their values that were used in this study.

4.1.4 Experimental Design

This section gives a quick high-level look at the experimental design methodology used in evaluating our algorithms and datasets. Strictly speaking, this experiment is of a factorial or fully-crossed design (a *8x3x3x6x6* factorial design). The exact factors and their values are listed in table 4.5. The product of the values in the last column is the total number of treatments (8x3x6x6 = 2,592). To minimize artifacts from the random nature of some of the algorithms, each treatment was repeated 10 times bringing the total number of passes to 25,920 and creating a 10-fold-crossed, method of evaluation.

Factor	**Levels**	**Level count**
dataset	ap203, ap214, BBC, BBCSport, ngBias3, ngBias8, ngBal3, and ngBal8	8
clusterer	KMeans, Canopy, and Genic	3
reduction method	PCA, FastMap, and TF-IDF Ranking	3
k	3, 5, 8, 15, 40, and 75	6
d	3, 15, 25, 50, 100, 200	6

Table 4.5: Experimental factors and their levels used for the experiment. A total of 2,592 (8x3x3x6x6) different treatments or combination of factors exist in this design

4.1.5 Workflow

This section lists the steps taken in the processing and evaluating the datasets in a chronological order.

1. **Conversion to ARFF**

   Convert the datasets into a congruent format. The supervised datasets were converted from the matrix market format (MTX) [24] while the unsupervised, STEP datasets were converted from an internal XML representation used within Hamlet (section 3.1).

2. **Dimensionality Reduction**

   At this point, each dataset was reduced 180 times (10 repeats times 3 reduction methods

times 6 values for dimensionality). These reduced datasets and a log of the time taken to perform the reduction were archived to disk.

3. **Clustering**

   Now that the data has been reduced and run-times for the reduction have been logged, clustering can take place. In this step, for each dataset, one sample from each reduction treatment (or each combination of dimensionality and reduction method) is ran through a clustering treatment (or each combination of number of clusters and clustering method) and repeated 10 times. This gives a total of 3,240 samples for each dataset, 25,920 in all. The resulting clusterings and their run-times are then archived to disk for later use.

4. **Analysis**

   This step can be performed at any time after a clustering has been achieved. This step includes gathering evaluation statistics on each of the 25,920 samples generated in the previous step. Here is where the metrics described in section 4.1.3 are collected.

4.2 Results

4.2.1 Dimensionality Reduction

Run-time

Unsurprisingly, run-times for dimensionality reduction were in line with each algorithm's expectations. Looking at the Mann-Whitney U tests in figure 4.6, you can see that reduction using tfidf was by far the fastest as it wins in both comparisons. FastMap comes in behind tfidf while losing to it and beating PCA. Lastly PCA brings up the rear losing in all tests. The MWU tests confirm that heuristic methods do have better run-times than PCA, but at what magnitude? The results in figure 4.7 show the run-times of each dimension reduction methods relative to PCA. These values

Reduction Method	ties	wins	losses	wins-losses
tfidf	0	2	0	2
FastMap	0	1	1	0
PCA	0	0	2	-2

Table 4.6: Mann-Whitney U tests at a 99% confidence level. As expected, the quick linear time method $tfidf$ performs better than both PCA and $FastMap$ while PCA performs the worst.

Reduction Method	Run-time relative to PCA
tfidf	<1
FastMap	3
PCA	100

Table 4.7: The above table illustrates the drastic computational requirements of PCA compared to FastMap and tfidf. tfidf runs in less than 1% of the time of PCA.

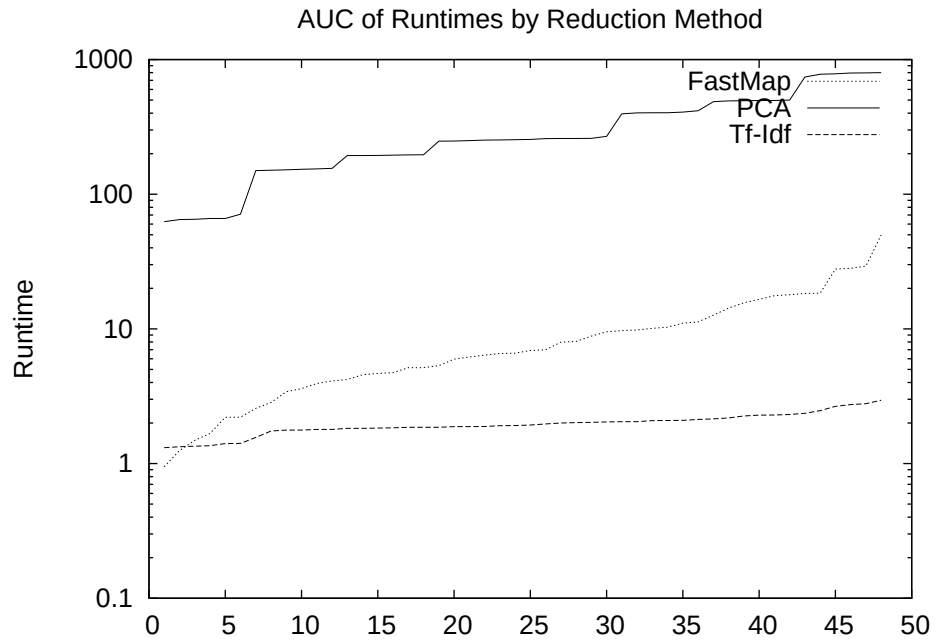


Table 4.8: Long Description

were taken by summing the results from every trial and then normalizing to a scale ranging from 0 to 100. tfidf shows an amazing performance benefit over PCA with its trials taking less than 1% of the time of PCA. FastMap also shows substantial performance relative to PCA taking only 3% of the time. These results can also be noted in figure 4.8 where the run-times have been sorted and plotted as a line.

As described in section 4.1.3, the three metrics being used to evaluate the validity of the results are purity, internal similarity, and external similarity. This section will discuss each of the results of each of these metrics. Additionally, we'll be looking at plots of the data points in three dimensions to get a graphical approximation of the types of structures generated by each of the reduction methods.

Reduction to Three Dimensions (A Visualization)

The plots in figure 4.2.1 show the visual representation of dimensionality reduction down to three dimensions in the STEP datasets. One thing to remember here is the drastic change in dimensionality this represents. The original dimensionality of these datasets can be as high as 15,000 (the datasets shown in figure 4.2.1 have around 1,000 and 3,000), so a drastic drop down to only 3 leaves plenty of room for data loss. Despite this caveat, PCA still provides a noticeable amount of detail in its three dimensional reduction compared to FastMap and tfidf. Notice the more random distribution of points in PCA plots. The FastMap points are limited to only three lines within the reduced space, while nearly all of the tfidf points are located in the same exact point. Realistically speaking, reduction to three dimensions is not practical for most IR tasks in textual domains. This would only be possible in an extremely small dataset of trivial use. This explains the results of the tfidf plots where the data is being reduced to only three terms. Any document that does not have those terms, will be placed at the same point. While this visualization does provide insight into the types of geometric structures created by these algorithms, it is still merely a visualization. Statistical tests are required before more concrete conclusions can be reached.

Cluster Purity

Unsurprisingly, PCA, the slow but more accurate algorithm, wins over both FastMap and tfidf and FastMap beats out tfidf which is again not a surprise. What is a surprise is the relatively small amount that tfidf and FastMap lose by. Both of these algorithms tie at only 83% of the purity of PCA, meaning that are relatively the same compared to PCA. Given the run-time results list above, a loss of 17% of accuracy is a small price to pay for a loss of 97% of CPU cycles. Furthermore, by looking the graph in 4.23 you can see nearly identical results out of all three methods for a majority of the time. With respect to cluster purity, little is lost when using these less robust but much faster alternatives.

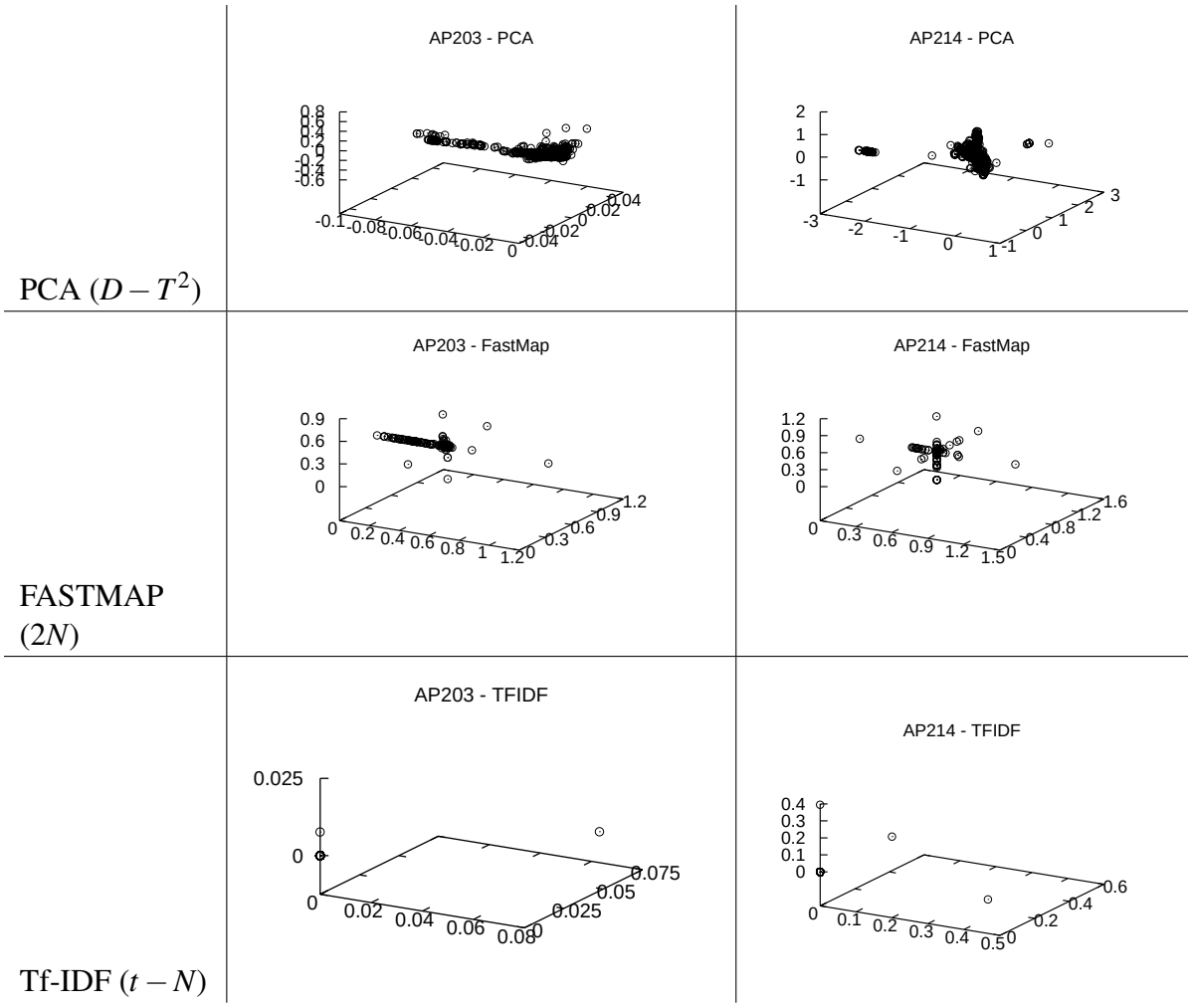


Figure 4.4: What is lost by heuristic exploration.

External Similarity

The results for external similarity have an even more interesting outcome. What makes these results interesting is the comparison to PCA. PCA, as expected, performs slightly better than FastMap, however tfidf does even better by coming in at 19% better than that. This order of results fits somewhat in line with amount of homogeneity each algorithm introduces into the new datasets. With FastMap, resulting document vectors are generally very similar, the same goes for PCA (at a lesser degree). With tfidf, we are actually introducing more variance in the dataset by removing the most frequent co-occuring terms.

Reduction Method	ties	wins	losses	wins-losses
PCA	0	2	0	2
TfIdf-Sort	0	1	1	0
Fastmap	0	0	2	-2

Table 4.9: Mann-Whitney U tests at a 95% confidence level.

Reduction Method	Purity relative to PCA
pca	100
fastmap	83
tfidf	83

Table 4.10:

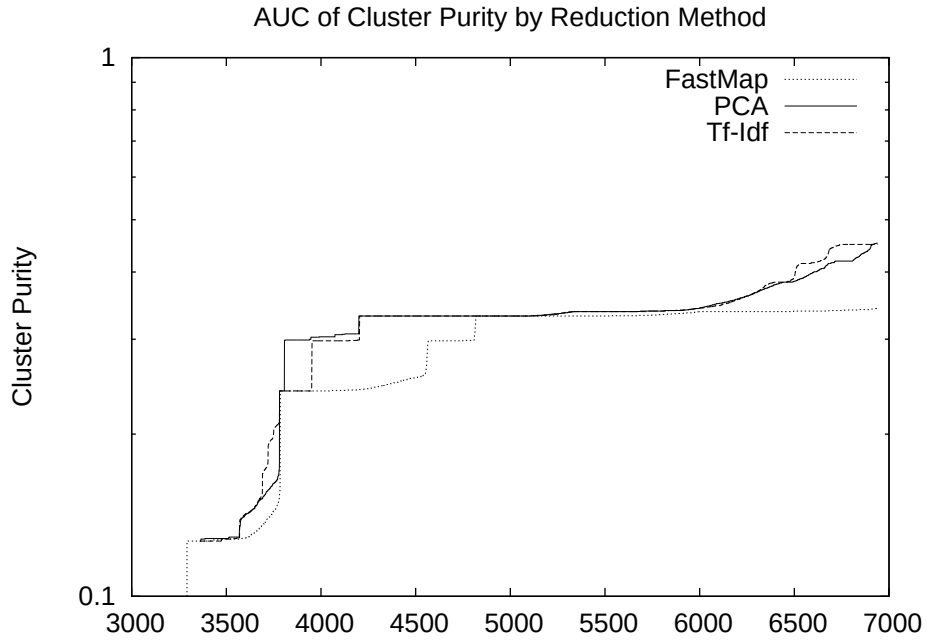


Table 4.11: Long Description

Internal Similarity

With internal ssimilarity we see FastMap coming in at 18% higher than PCA. This is most likely attributed to the nature of FastMap's approach. With each incremental step of the FastMap algorithm, the data becomes increasingly uniform. Looking at the plots of FastMap-reduced data in 4.2.1, you'll notice that even though the data lives in a three dimensional space, it's a three dimensional space that is largely segregated into symmetric planes. For the most rudimentary of all three algorithms tfidf comes in at only 88% of the internal similarity of PCA.

Similarity Loss

The combined similarity scores for dimension reduction confirm the baseline idea that PCA outperforms both FastMap and tfidf. Despite FastMap having an 18% lead with respect to internal

Reduction Method	ties	wins	losses	wins-losses
pca	0	2	0	2
tfidf-sort	1	0	1	-1
fastmap	1	0	1	-1

Table 4.12: Mann-Whitney U tests at a 95% confidence level.

Reduction Method	External similarity relative to PCA
fastmap	102
pca	100
tfidf	81

Table 4.13:

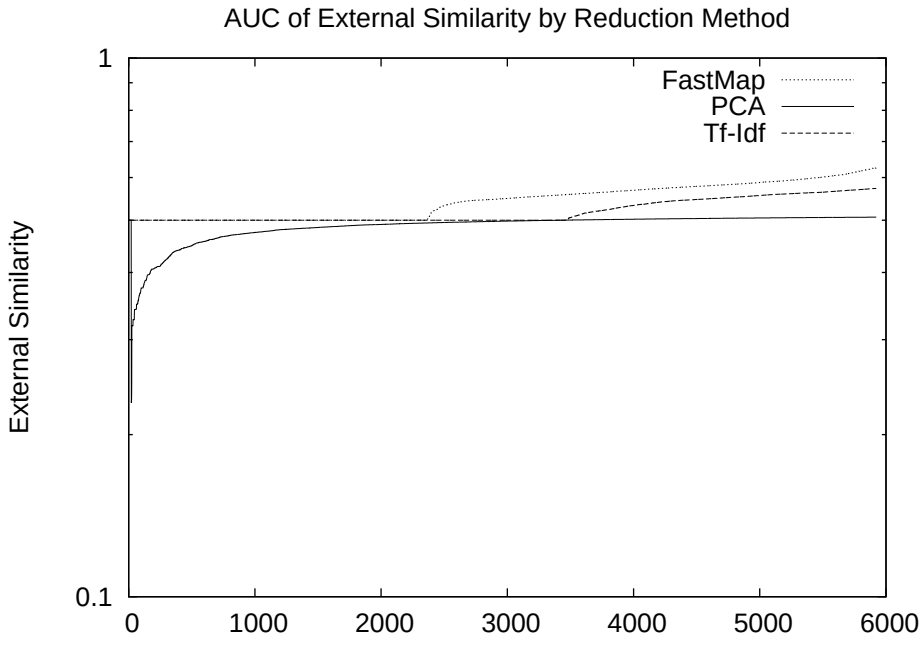


Table 4.14: Long Description

Reduction Method	Similarity Loss
tfidf	6
fastmap	30

Figure 4.5: Comparative loss of combined similarity by reduction method

similarity, its still 48% worse at external similarity. tfidf on the other hand, though still not outperforming PCA, only lost 6% collective similarity. Which, when given its massive run-times could very well be enough to swallow.

Reduction Method	ties	wins	losses	wins-losses
pca	0	2	0	2
fastmap	0	1	1	0
tfidf-sort	0	0	2	-2

Table 4.15: Mann-Whitney U tests at a 95% confidence level.

Reduction Method	Internal similarity relative to PCA
fastmap	118
pca	100
tfidf	88

Table 4.16:

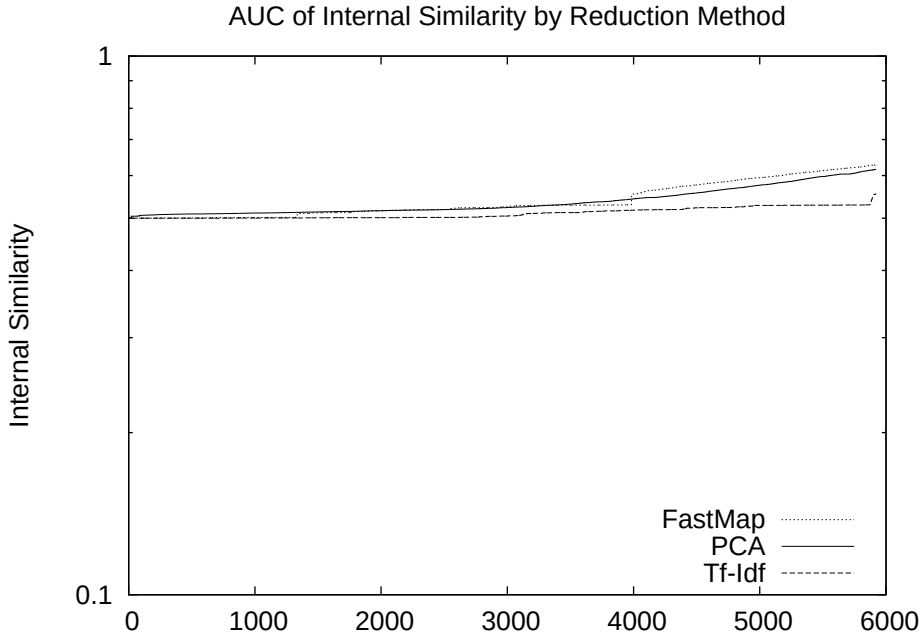


Table 4.17: Long Description

4.2.2 Clustering

Run-time

The run-time results for clustering are not as staggering as those of dimension reduction, but still very impressive. Performing the canopy optimization to K-Means has cut run-times to 1/2 of vanilla K-Means. Even more impressive than this is the fact that Genic was running at approximately 6% of K-Means' time. Genic uses a linear time approach for its clustering mechanism and these run-time scores certainly illustrate this point. K-Means' run-time on the other hand is variable and has no known theoretical upper bounds. In some cases, if not given an appropriate escape criteria, the algorithm can continue on infinitely.

Clustering Method	ties	wins	losses	wins-losses
genic	0	2	0	2
kmeans	1	0	1	-1
canopy	1	0	1	-1

Table 4.18: Mann-Whitney U tests at a 95% confidence level.

Clustering Method	Run-Time Relative To KMeans
genic	6
canopy	52
kmeans	100

Table 4.19:

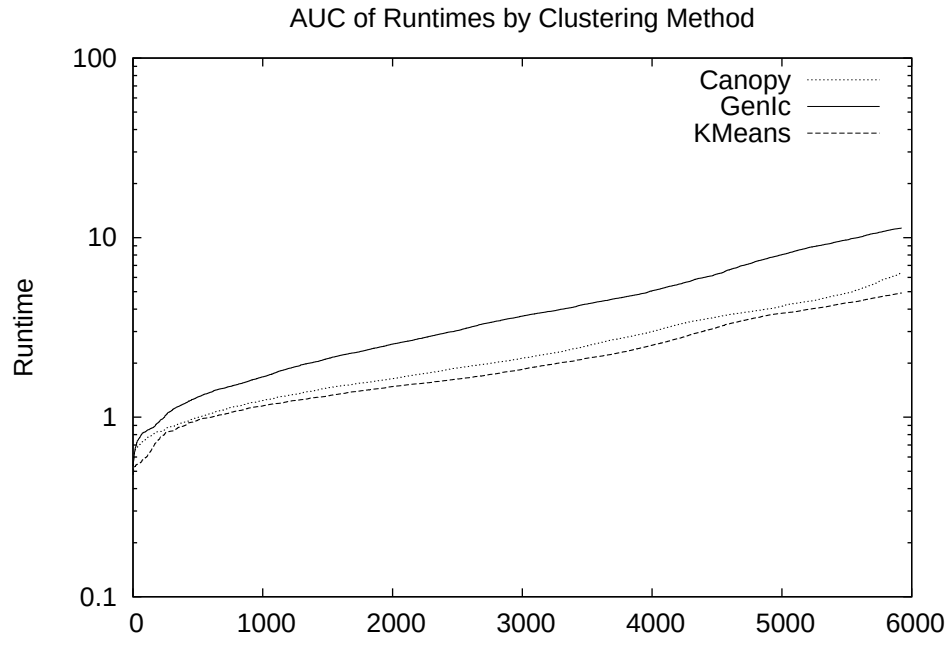


Table 4.20: Long Description

Cluster Purity

As expected, given the drastic performance increases, the accuracy of these heuristic algorithms does suffer. With the fastest, Genic, suffering the most at only 64% of the KMeans purity. The most likely cause for the lack of purity with the Genic results my be attributed to its generational update phase. In this phase, Genic will compute probabilities of survival for each potential cluster center, removing those with probabilities below the threshold. As a result, there are many times when the number of clusters returned is far less than both the number requested and the intrinsic number existing in the dataset. Canopy clustering performed only slightly better than Genic coming in at 72% of KMeans. Though, given its 50% cut in run-time, there is some consideration to be given to a 28% cut in purity.

Clustering Method	ties	wins	losses	wins-losses
kmeans	0	2	0	2
genic	0	1	1	0
canopy	0	0	2	-2

Table 4.21: Mann-Whitney U tests at a 95% confidence level.

Clustering Method	Cluster Purity Relative To KMeans
kmeans	100
canopy	72
genic	64

Table 4.22:

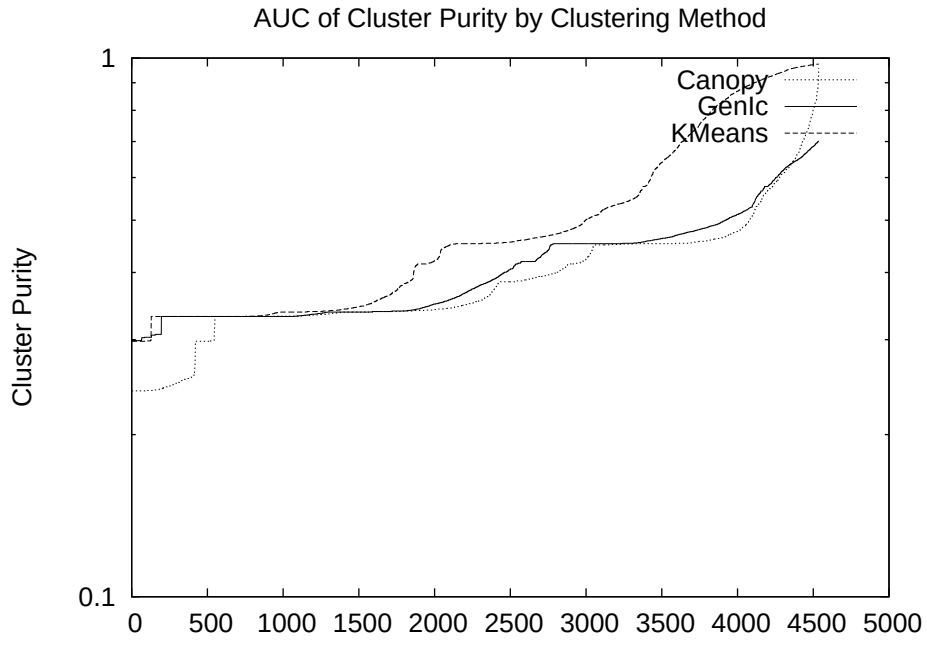


Table 4.23: Long Description

External Similarity

With external similarity, we see a pattern similar to that of purity. We have Genic, the fastest algorithm providing the least in terms of resulting accuracy, though not nearly as bad as with purity. We also see Canopy coming in ahead of Genic, as opposed to the case of external similarity, Genic actually outperforms KMeans. External similarity is a measure of similarity between things in different clusters so in an ideal clustering this number is low. In this example Canopy comes in 9% ahead of KMeans. A 9% increase may appear to only be marginally better than KMeans, however this is not the case. The results in 4.24 show the MWU test results of the same data. Genic beats both KMeans and Canopy with statistical significance at a 95% confidence level.

Clustering Method	ties	wins	losses	wins-losses
genic	0	2	0	2
kmeans	0	1	1	0
canopy	0	0	2	-2

Table 4.24: Mann-Whitney U tests at a 95% confidence level.

Clustering Method	Cluster External Similarity Relative To KMeans
genic	91
kmeans	100
canopy	113

Table 4.25:

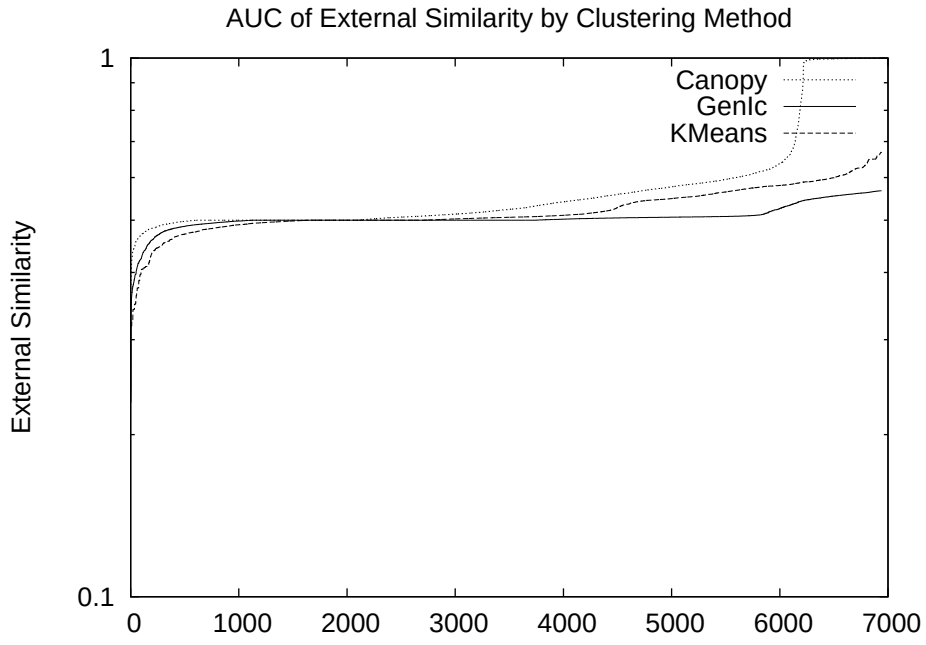


Table 4.26: Long Description

Internal Similarity

With internal similarity, neither Canopy or Genic can compete with KMeans. Genic comes closest with only a 9% loss and Canopy is not too far off with 83% loss. It makes sense that we have Canopy clustering, the algorithm which performed the best in terms of *external similarity*, performing the worst in *internal similarity*. This tells us much about the resulting clusters produced by the Canopy algorithm. Specifically, that these clusters have a tendency to contain items with more inherent difference than either of the other two algorithms.

Clustering Method	ties	wins	losses	wins-losses
kmeans	0	2	0	2
genic	0	1	1	0
canopy	0	0	2	-2

Table 4.27: Mann-Whitney U tests at a 95% confidence level.

Clustering Method	Cluster Internal Similarity Relative To KMeans
canopy	83
genic	91
kmeans	100

Table 4.28:

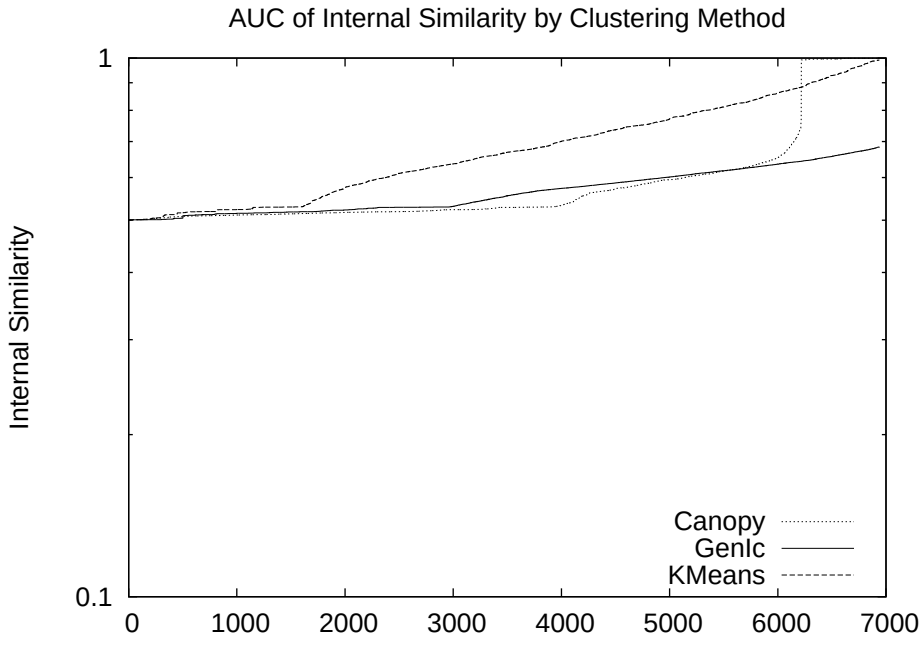


Table 4.29: Long Description

Clustering Method	Similarity Loss
canopy	26
genic	22

Figure 4.6: Comparative loss of combined similarity by clustering method

Similarity Loss

The combined loss of similarity for clustering algorithms is shown above in 4.6. One interesting bit of information taken from this is that despite Canopy's improved performance with respect to external similarity, its lesser performance on internal similarity actually ranks it below Genic, though not by much. This fact certainly has an effect on the performance of Genic related to the other two algorithms, both of which consistently return the value of k requested. While the impact of this is hard to discern from the data on hand, this fact will be considered when assessing Genic

as a viable heuristic alternative.

Genic's Variable Number of Clusters

Size = k	Average k actually returned	Percent of k
1	1	100
4	3.9375	98
16	11.7875	74
32	20.275	63
64	35.5375	56
128	64.4375	59
256	114.013	45
512	189.15	37
1024	316.775	31

Figure 4.7: Average number of clusters returned by GENIC for each K

An interesting concern was noted when looking at the clustering results for Genic. The data above in 4.7 shows the large variation in number of clusters specified by the parameter k and the actual number of clusters returned. The percentage of k that is actually returned as the number of clusters has an inverse relationship to k.

4.2.3 Combining Methods

Up until now we have examined both dimension reduction and clustering, though only in isolation. The results for dimensionality reduction were produced from analysis of all clustered results segmented by the reduction algorithm and vice versa for the clustering results. While this has provided us further insight into the behavior of each algorithm on its own, the true value is gained in combining heuristic approaches for dimensionality reduction and clustering. This is explored within this section.

Run-time

The results of run-time presented in 4.32 show, among other things, an interesting trend with clusterers paired with PCA. All treatments that include PCA live at the bottom of the run-time

Reducer-Clusterer	ties	wins	losses	wins-losses
tfidf-genic	0	8	0	8
tfidf-kmeans	0	7	1	6
fastmap-genic	0	6	2	4
fastmap-kmeans	0	5	3	2
tfidf-canopy	0	4	4	0
fastmap-canopy	0	3	5	-2
pca-genic	0	2	6	-4
pca-canopy	0	1	7	-6
pca-kmeans	0	0	8	-8

Table 4.30: Mann-Whitney U tests at a 95% confidence level.

Reducer-Clusterer	Run-time Relative To pca-kmeans
tfidf-genic	1
fastmap-genic	3
tfidf-canopy	5
fastmap-canopy	7
tfidf-kmeans	12
fastmap-kmeans	20
pca-genic	71
pca-canopy	76
pca-kmeans	100

Table 4.31:

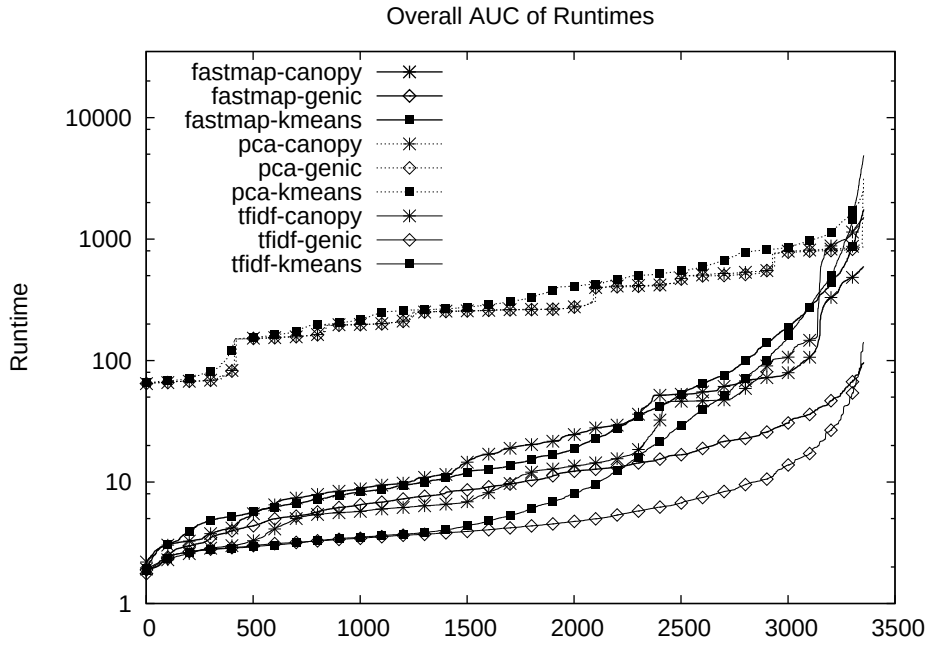


Table 4.32: Long Description

performance scale. Anything that is not using PCA lives at the top, with all treatments performing at less than 20% of the $pca-kmeans$ treatment. This tells that the bulk of CPU cycles of the baseline treatment, $pca-kmeans$, is coming from PCA. This is an interesting discovery because it shows that even if KMeans is used for clustering, great run-time benefits can come from simply using an alternative to PCA for dimensionality reduction. This is further illustrated by the MWU results in 4.30. In this figure both non-PCA KMeans treatments ($fastmap-kmeans$ and $tfidf-kmeans$) can be seen outperforming the same non-PCA treatments of Canopy clustering.

Other interesting trends to note in 4.32 are the two non-PCA genic treatments which dominate

the area of the chart for lower run-times. The tfidf-genic treatment in particular comes in at only 1% of pca-kmeans. Aside from these two genic treatments, all others live in roughly the same region which the specific ranking denoted in 4.30.

Cluster Purity

Reducer-Clusterer	ties	wins	losses	wins-losses
pca-kmeans	0	8	0	8
pca-genic	0	7	1	6
fastmap-genic	1	5	2	3
fastmap-canopy	1	5	2	3
tfidf-kmeans	1	3	4	-1
fastmap-kmeans	1	3	4	-1
tfidf-canopy	0	2	6	-4
tfidf-genic	0	1	7	-6
pca-canopy	0	0	8	-8

Table 4.33: Mann-Whitney U tests at a 95% confidence level.

Reducer-Clusterer	Purity Relative To pca-kmeans
pca-kmeans	100
pca-genic	95
tfidf-kmeans	73
fastmap-kmeans	73
tfidf-genic	67
fastmap-genic	68
tfidf-canopy	50
fastmap-canopy	50
pca-canopy	38

Table 4.34:

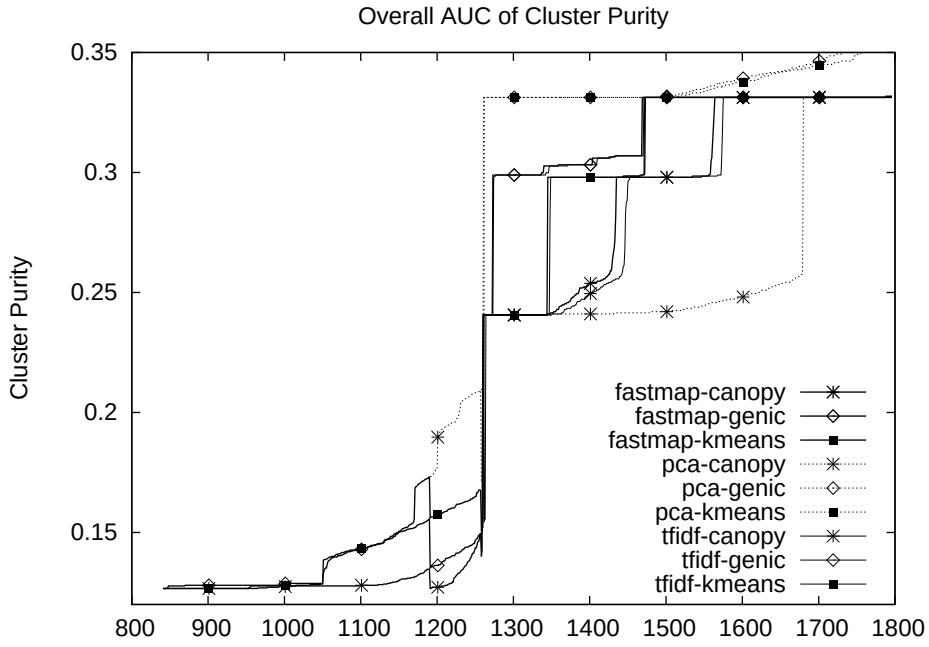


Table 4.35: Long Description

With respect to cluster purity, nearly all of the results are split along choice of clusterer. In 4.35 notice the dichotomy between clusterers. All of the lower scoring treatments along the right side of the graph belong to Canopy. Apart from the pca-canopy treatment, the other Fastmap and TfIdf

canopy treatments follow the exact pattern of purity. The same is true for KMeans and Genic. Both of their Fastmap and TfIdf scores following the same trend.

Perhaps what is most interesting about this graph is the alignment of KMeans and Genic. In the case of the PCA pairing of KMeans and Genic, they following nearly the identical pattern. With Fastmap and TfIdf the same pattern occurs. Even these two separate reduction methods follow a similar shape when paired with KMeans. This is counterintuitive. Given the close algorithmic relationship between Canopy and KMeans, these would seem to be the most likely pair, not Genic and KMeans.

External Similarity

With external similarity we have five of the treatments performing better than the baseline of pca-kmeans, with all three of the Canopy treatments grouped into this category. Though, this is coming from the relative median list in table 4.37 and is not entirely as it may seem. The MWU results in 4.36 paint a clearer picture though we still have the pair, tfidf-genic, outperforming the baseline.

Several patterns can be seen emerging in 4.38. The most prominent patter being the large amount of treatments starting around 0.5 and then breaking off at various points with a relatively similar acceleration. Each of these breaking off points appear to be equally spaced which likely indicates that one of our variables is is causing this. Probably causes for these regular divergences are k (number of clusters), d (number of dimensions reduced to), and variations in each dataset. Looking at the external similarities for clusterers and reducers on their own (figure 4.2.3) we notice the same regulated divergences. At first this may seem to indicate an issue of different datasets. Though this may be one contributing factor, a closer look at the two side-by-side shows that the reduction method seems to have the stronger pull. If this is the case, the divergence is likely attributed to variations of d which are not conducive to the particular algorithm performing well on a given dataset.

Reducer-Clusterer	ties	wins	losses	wins-losses
pca-kmeans	0	8	0	8
tfidf-genic	0	7	1	6
pca-genic	0	6	2	4
tfidf-kmeans	1	4	3	1
pca-canopy	0	4	4	0
tfidf-canopy	1	3	4	-1
fastmap-genic	0	2	6	-4
fastmap-kmeans	0	1	7	-6
fastmap-canopy	0	0	8	-8

Table 4.36: Mann-Whitney U tests at a 95% confidence level.

Reducer-Clusterer	External Similarity Relative To pca-kmeans
tfidf-canopy	78
fastmap-canopy	79
tfidf-kmeans	80
pca-canopy	81
fastmap-kmeans	87
pca-kmeans	100
pca-genic	102
tfidf-genic	103
fastmap-genic	107

Table 4.37:

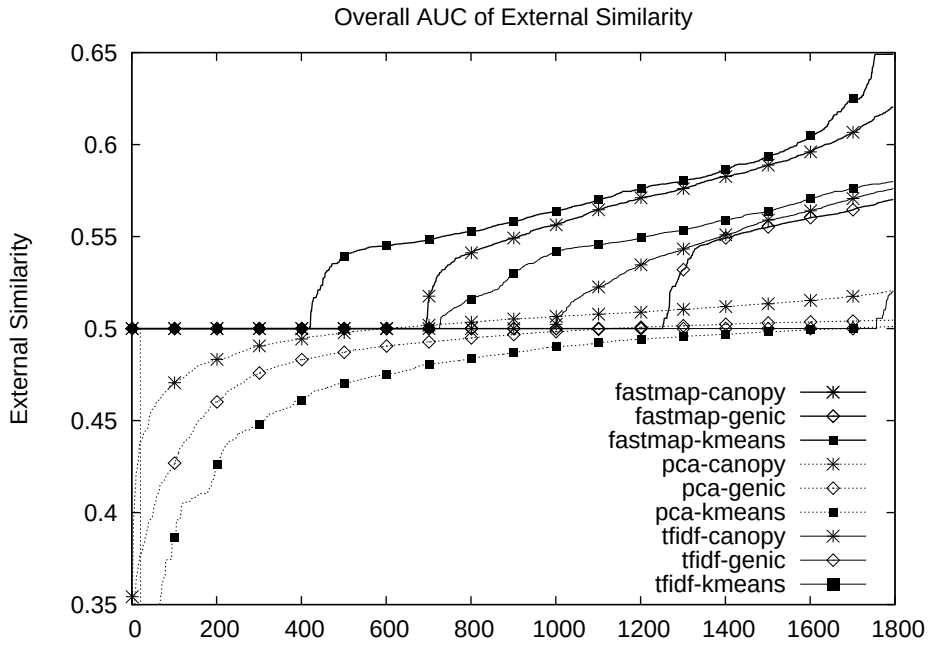


Table 4.38: Long Description

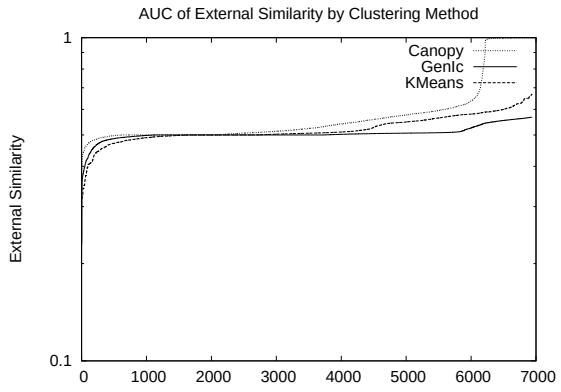


Table 4.39: External Similarity by Clusterer

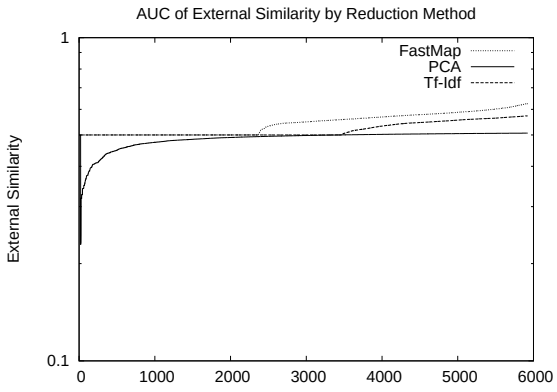


Table 4.40: External Similarity by Reduction Method

Internal Similarity

Reducer-Clusterer	ties	wins	losses	wins-losses
pca-kmeans	0	8	0	8
pca-genic	0	7	1	6
fastmap-genic	1	5	2	3
fastmap-canopy	1	5	2	3
tfidf-kmeans	1	3	4	-1
fastmap-kmeans	1	3	4	-1
tfidf-canopy	0	2	6	-4
tfidf-genic	0	1	7	-6
pca-canopy	0	0	8	-8

Table 4.41: Mann-Whitney U tests at a 95% confidence level.

Reducer-Clusterer	Internal Similarity Relative To pca-kmeans
pca-kmeans	100
fastmap-kmeans	90
pca-genic	88
fastmap-genic	82
fastmap-canopy	81
tfidf-kmeans	79
tfidf-genic	75
tfidf-canopy	74
pca-canopy	71

Table 4.42:

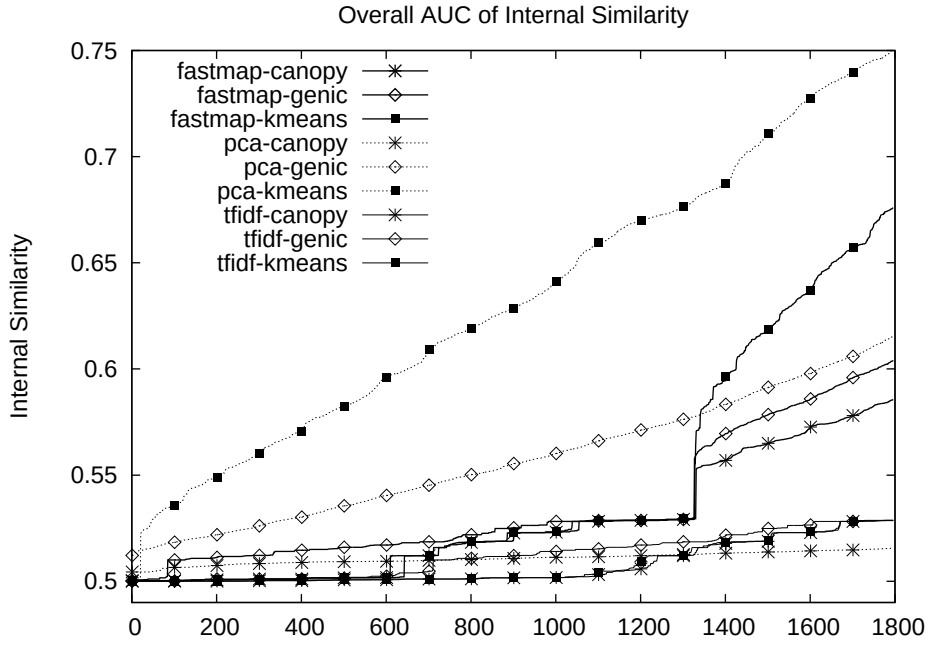


Table 4.43: Long Description

The most surprising aspect of combined internal similarity is the giant disparity in some of the PCA treatments. For instance, $pca-kmeans$ and $pca-genic$ perform at the top of the MWU results in table 4.41 while $pca-canopy$ is at the very bottom with 8 losses. Apart from external similarity, looking back at every other comparison assessing the accuracy of Canopy clustering you'll notice a definite trend in its under-performance. In the case of combined internal similarity, this under-performance of Canopy is enough to bring down the best performing dimensionality reduction method, PCA. A similar pattern can be seen with the Fastmap results, only the disparity

does not begin until the higher similarity values are approached. In both the Fastmap and PCA results we have a three way divergence between KMeans, Genic, and Canopy spreading out with different rates of increase. In both cases, the same order applies, KMeans -¿ Genic -¿ Canopy.

Right beneath the $pca - kmeans$ baseline are all Fastmap treatments. In section 4.2.1, we noticed the same pattern and attributed it to the homogenistic nature of the Fastmap algorithm and it appears to be prevalent in this analysis as well.

Similarity Loss

Reducer-Clusterer	**Similarity Loss Relative To pca-kmeans**
fastmap-kmeans	-3
fastmap-canopy	-2
tfidf-kmeans	1
tfidf-canopy	4
pca-canopy	10
pca-genic	14
fastmap-genic	25
tfidf-genic	28

Figure 4.8: Reducer-Clusterer Relative Similarity Loss

Reducer-Clusterer	**ties**	**wins**	**losses**	**wins-losses**
pca-kmeans	0	8	0	8
tfidf-kmeans	1	6	1	5
fastmap-kmeans	1	6	1	5
pca-genic	0	5	3	2
tfidf-canopy	2	2	4	-2
fastmap-genic	2	2	4	-2
fastmap-canopy	2	2	4	-2
tfidf-genic	0	1	7	-6
pca-canopy	0	0	8	-8

Figure 4.9: Reducer-Clusterer Similarity Difference MWU tests at a 95% confidence level.

The results of similarity loss are displayed in two forms for combined reduction and clustering. The first is the same table used for clustering and reduction on their own. The second is a table of

MWU results derived from the difference of the actual external and internal similarity values. Since similarity loss implies a comparison to a baseline, the MWU table is not a display of similarity loss but rather a display of similarity difference.

The fact that Fastmap lives on top of table 4.8 is an illustration of homogeneously reduced data being clustered effectively with both Canopy and KMeans clustering. Directly below Fastmap in table 4.8 is the same two clusterers, only in this case being reduced by TfIdf, the ”straw man” reduction algorithm. Both FastMap and TfIdf product highly homogeneous results which shows further that a well-separated dataset may not necessarily be mandatory for fast and accurate clustering.

Looking at the data in table 4.8, we can see that there are two treatments whose loss in one similarity is offset by a gain in the other. Though the magnitude of these values are small, what they imply is much greater. These difference show that algorithms with drastic reductions in runtime are still capable of producing results near or better than their rigorous alternatives.

The MWU results in table 4.9 have all KMeans treatments ranking highest. This could either be an example of how strong of a clustering algorithm that KMeans is or rather, an indication the ineffectiveness of the more rigorous reduction method, PCA. Looking at the win-loss-tie of $pca - canopy$ and the fact that it lies on the bottom shows more insight. With both the TfIdf and Fastmap pairings of Canopy performing better than the PCA pairing, there exists further proof that PCA may not be necessary.

Chapter 5

Conclusions

5.1 Specific Findings

5.1.1 Dimension Reduction

When looking the validity of the dimensionality reduction algorithms examined in this study, its easily seen that our rigorous algorithm, PCA, wins in all metrics, hands down. Not one of the heuristic approaches beats the purity, external similarity, and internal similarity of the PCA results. However, this does not mean that they are far off the from PCA results. While Fastmap does perform worst in terms of the MWU of the solo purity measure, it beats TfIdf in internal similarity and ties in external similarity. What is most important for Fastmap though, is its performance in the combined results. With the exception of external similarity, Fastmap outperforms TfIdf and comes close to the baseline set by PCA. Because of all this, in situations where external similarity is not as important as purity or internal similarity, that is, situations that require items within the same clustering to be of higher similarity, Fastmap is our recommended approach. This of course is assuming that the run-time of the given system is priority and the long run-times of PCA are not acceptable. That being said, the TfIdf results were quite impressive given its naivety and the fact that it was created in-house as a straw man benchmark. Owing to naivety, TfIdf's run-times are

	Reduction		Clustering		Combined	
	treatment	w-l	treatment	w-l	treatment	w-l
Purity	PCA	2	kmeans	2	pca-kmeans	8
	tfidf	0	genic	0	pca-genic	6
	fastmap	-2	canopy	-2	fastmap-genic	3
					fastmap-canopy	3
					tfidf-kmeans	-1
					fastmap-kmeans	-1
					tfidf-canopy	-4
					tfdidf-genic	-6
					pca-canopy	-8
External Similarity	PCA	2	genic	2	pca-kmeans	8
	tfidf	-1	kmeans	0	tfidf-genic	6
	fastmap	-1	canopy	-2	pca-genic	4
					tfidf-kmeans	1
					pca-canopy	0
					tfidf-canopy	-1
					fastmap-genic	-4
					fastmap-kmeans	-6
					fastmap-canopy	-8
Internal Similarity	PCA	2	kmeans	2	pca-kmeans	8
	fastmap	0	genic	0	pca-genic	6
	tfidf	-2	canopy	-2	fastmap-genic	3
					fastmap-canopy	3
					tfidf-kmeans	-1
					fastmap-kmeans	-1
					tfidf-canopy	-4
					tfidf-genic	-6
					pca-canopy	-8
Run-time	tfidf	2	genic	2	tfidf-genic	8
	fastmap	0	kmeans	-1	tfidf-kmeans	6
	PCA	-2	canopy	-1	fastmap-genic	4
					fastmap-kmeans	2
					tfidf-canopy	0
					fastmap-canopy	-2
					pca-genic	-4
					pca-canopy	-6
					pca-kmeans	-8

Table 5.1: An aggregate look at all MWU results broken down by category

Reducer	Clusterer	Relative Run-time
tfidf	*	<1%
fastmap	*	3%
PCA	*	100%
*	genic	6%
*	kmeans	52%
*	canopy	100%
tfidf	genic	1%
tfidf	kmeans	12%
fastmap	genic	3%
fastmap	kmeans	20%
tfidf	canopy	5%
fastmap	canopy	7%
pca	genic	71%
pca	canopy	76%
pca	kmeans	100%

Table 5.2: An aggregate look at all AUC sums relative to PCA, Kmeans, or PCA-Kmeans

nearly linear at $d * log(d)$ (with d equal to the number of dimensions before reduction). In addition to this, its validity scores are not far off from Fastmap. Even TfIdf may prove acceptable if the system finds that the Fastmap run-times leave a little more to be desired.

5.1.2 Clustering

Our clustering conclusions are similar to our dimensionality reduction results in that the rigorous approach outperformed the alternatives in most measurements of validity. However, the clustering results differ from the dimensionality reduction results in that there is a clear leader among the two heuristic approaches. Genic beats Canopy in every validity measure and even outperforms *Kmeans* when it comes to external similarity. This says quite a lot for an algorithm that is not nearly as well known in the program comprehension arena. Canopy on the other hand, is at the very bottom of the spectrum. This is likely one of the most surprising conclusions of the study given the close relationship of Canopy and KMeans. These results were verified by several people on several occasions given that Canopy clustering is generally expected to perform close to, or at the same level of KMeans in terms of the validity of its resulting clusters and that these results clearly contradict that. Because of all of the above factors, the conclusion of this study with respect to clustering is that while Genic is not at the level of KMeans, it can be close enough when the run-time of KMeans is too much to bear.

5.1.3 Effective Combinations

Drawing from the conclusions listed in the two sections above, we have found that using Fastmap for dimensionality reduction combined with Genic for clustering can produce effective results in a fraction of the time. That being said, even more potent clusters can be achieved by simply replacing only PCA. The TfIdf and KMeans combination shows astounding performance in validity at only 12% of the base line run-time. The Fastmap and KMeans combination also shows solid performance at only 20% of the PCA and KMeans run-time.

5.2 Summary

To summarize the entirety of this work in one statement; heuristic methods are worth it. That being said, there are a few caveats that exist and this section aims to identify those. One thing that was learned in this study is that a complete replacement of a combined PCA and KMeans solution with an entirely heuristic solution shows incredible benefits to performance. However, using a solely heuristic approach may lead to sacrifices in validity. Although these sacrifices are minimal compared to the increased performance, they might not be acceptable in every application. In order to fully optimize the speed/accuracy ratio, you must consider your run-time constraints as well as how much inaccuracy your application can tolerate. It is my opinion though that using the right combinations or heuristic and rigorous clusterers and reducers can lead to acceptable losses in validity in most situations.

5.3 Future Work

- Include more experiments that more closely mimic the in-use applications of program comprehension
- Perform more in-depth analysis on the effect of the different parameters of each of the algorithms
- Incorporate more current algorithms such as Latent Semantic Indexing to give the study more breadth over what is happening in the industry
- Explorations into domains where Genic could potentially provide run-times not before seen of clustering algorithms.
- A look into different approaches of determining the inherent dimensionality of datasets and how this can affect clustering performance

Bibliography

[1] Khurshid Ahmad, Bogdan Vrusias, and Paulo C. F. de Oliveira. Summary evaluation and text categorization. In *SIGIR2003*, pages 443–444, 2003.

[2] Machine Learning Group at UCD. Mlg datasets, Jan 2009. `http://mlg.ucd.ie/datasets`.

[3] Stephen D. Bay, Dennis F. Kibler, Michael J. Pazzani, and Padhraic Smyth. The UCI KDD archive of large data sets for data mining research and experimentation. *SIGKDD Explorations*, 2(2):81–85, 2000.

[4] Davor Cubranic, Gail C. Murphy, Janice Singer, and Kellogg S. Booth. Hipikat: A project memory for software development. *IEEE Transactions on Software Engineering*, 31(6):446–465, 2005.

[5] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.

[6] Chris Ding and Xiaofeng He. K-means clustering via principal component analysis. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 29, New York, NY, USA, 2004. ACM.

[7] Christos Faloutsos and King-Ip Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In Mchael Carey and Donovan Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174. ACM Press, 1995.

[8] Gregory Gay, Tim Menzies, Bojan Cukic, and Burak Turhan. How to build repeatable experiments. In *PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, pages 1–9, New York, NY, USA, 2009. ACM.

[9] Derek Greene and Pádraig Cunningham. Practical solutions to the problem of diagonal dominance in kernel document clustering. In *Proc. 23rd International Conference on Machine learning (ICML'06)*, pages 377–384. ACM Press, 2006.

[10] Seth Grimes. Unstructured data and the 80 percent rule. Experts Corner: Seth Grimes, Clarabridge Bridgepoints, Issue 3, 2008, Q3 2008. White Paper.

[11] F.A. Grootjen, D.C. van Leijenhorst, and Th. P. van der Weide. A formal derivation of heaps' law, 2003.

[12] Chetan Gupta and Robert Grossman. Genic: A single pass generalized incremental algorithm for clustering. In *In SIAM Int. Conf. on Data Mining*. SIAM, 2004.

[13] Hans Van Halteren, Jakub Zavrel, and Walter Daelemans. Improving accuracy in word-class tagging through combination of machine learning systems. *Computational Linguistics*, 27:2001, 2001.

[14] Jane Huffman Hayes, Alex Dekhtyar, and James Osborne. Improving requirements tracing via information retrieval. In *in Proceedings of the International Conference on Requirements Engineering (RE*, pages 151–161, 2003.

[15] ISO. *ISO 10303-11:1994: Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual*. pub-ISO, 1994.

[16] R. Jardim-Goncalves, N. Figay, and A. Steiger-Garcao. Enabling interoperability of step application protocols at meta-data and knowledge level. *International Journal of Technology Management*, pages 402–421, 2006.

[17] I. Jolliffe. *Principal component analysis. 2nd edition*. Springer, 2002.

[18] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. The analysis of a simple k-means clustering algorithm. In *UMD*, 2000.

[19] Ken Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339, 1995.

[20] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178, New York, NY, USA, 2000. ACM.

[21] Jian-Yun Nie, Michel Simard, Pierre Isabelle, and Richard Durand. Cross-language information retrieval based on parallel texts and automatic mining of parallel texts from the web. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 74–81, New York, NY, USA, 1999. ACM.

[22] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[23] Martin F. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.

[24] Karin A. Remington Ronald F. Boisvert, Roldan Pozo. The matrix market exchange formats: Initial design. *NISTIR*, 1997.

[25] D. E. Rose and R. K. Belew. Legal information retrieval a hybrid approach. In *ICAIL '89: Proceedings of the 2nd international conference on Artificial intelligence and law*, pages 138–146, New York, NY, USA, 1989. ACM.

[26] Horacio Saggion and Guy Lapalme. Concept identification and presentation in the context of technical text summarization. In *NAACL-ANLP 2000 Workshop on Automatic summarization*, pages 1–10, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[27] Gerard Salton. The smart document retrieval project. In *SIGIR '91: Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 356–358, New York, NY, USA, 1991. ACM.

[28] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.